Dynamically Delayed Postdictive Completeness and Consistency in Learning

John Case and Timo Kötzing*

July 4, 2008

Abstract

In computational function learning in the limit, an algorithmic *learner* tries to find a program for a computable function g given successively more values of g, each time outputting a conjectured program for g. A learner is called *postdictively complete* iff all available data is correctly postdicted by each conjecture.

Akama and Zeugmann presented, for each choice of natural number δ, a relaxation to postdictive completeness: each conjecture is required to postdict only all *except the last δ seen* data points.

This paper extends this notion of delayed postdictive completeness from *constant* delays to *dynamically* computed delays. On the one hand, the delays can be different for different data points. On the other hand, delays no longer need to be by a fixed finite number, but any type of computable countdown is allowed, including, for example, countdown in a system of ordinal notations and in other graphs disallowing *computable* infinitely descending counts.

We extend many of the theorems of Akama and Zeugmann and provide some feasible learnability results. Regarding *fairness* in feasible learning, one needs to limit use of tricks that postpone output hypotheses until there is enough time to "think" about them. We see, for polytime learning, postdictive completeness (and delayed variants): 1. allows *some* but *not* all postponement tricks, *and* 2. there is a surprisingly tight boundary, for polytime learning, between what postponement is allowed and what is not. For *example*: 1. the set of polytime computable functions *is* polytime postdictively completely learnable employing some postponement, *but* 2. the set of exptime computable functions, while polytime learnable with a little more postponement, is *not* polytime postdictively completely learnable! We have that, for w a notation for ω, the set of exptime functions *is* polytime learnable with *w-delayed* postdictive completeness. Also provided are generalizations to further, small constructive limit ordinals.

1 Introduction

"Explanatory learning", or Ex-learning for short, is a standard model of limit learning of computable functions. In this model a learner is given successively longer initial segments of a target function. For each initial segment of the target function, the learner gives an hypothesis. The learner is said to successfully Ex-*learn* the target function iff the infinite sequence of hypotheses generated by the learner on the initial segments of the target functions converges in the limit to a (single) correct program for the target function [JORS99].

In some literature on limit learning this intuitively simple success criterion is used as a minimal requirement for success, and additional requirements are defined and examined. We call two such extra requirements *postdictive completeness* (the hypotheses correctly postdict the data seen so far) and *postdictive consistency* (the hypotheses do not explicitly contradict the given data) [Bār74, BB75, Wie76, Wie78].[1] There are Ex-learnable sets of functions that cannot be learned with the additional requirement of postdictive completeness or consistency.

*Student author.

[1] We use the terminology *postdictive completeness* because the the hypotheses must *completely* postdict the data seen to that point. We use the terminology *postdictive consistency* because the the hypotheses need only *avoid explicit inconsistencies* with the data seen to that point. Such an hypothesis may, then, on some input for which the data seen to that point tells the answer, go undefined (i.e., go into an infinite loop) and, thereby, not explicitly contradict the known data. In the literature on these requirements, except for [Ful88], what we call postdictively complete is called *consistent*, and what we call postdictively consistent is called *conformal*.

Akama and Zeugmann [AZ07] presented success criteria that are a little less restrictive than postdictively complete Ex-learning. Their criteria delay the requirement to postdict a given datum by a fixed natural number δ of (not necessarily distinct) hypotheses output. For ordinary postdictive completeness, if a learner h has seen so far, on a computable g input, $g(0), \ldots, g(n-1)$, then h's corresponding hypothesis, p_n, must correctly compute $g(0), \ldots, g(n-1)$.[2] For delay δ, Akama and Zeugmann, require only that, on $g(0), \ldots, g(n-1)$, h's *later* hypothesis, $p_{n+\delta}$, must correctly compute $g(0), \ldots, g(n-1)$. Essentially, the delay δ learner could, after seeing $g(0), \ldots, g(n-1)$, run a counter down from δ to 0 to see which future hypothesis must correctly compute $g(0), \ldots, g(n-1)$.

In the present paper we extend this notion of delayed postdictive completeness from *constant* delays δ to *dynamically computed* delays. *One* of the ways we consider herein to do this involves counting down from notations for constructive ordinals. We explain. Everyone knows how to use the natural numbers for counting, including for counting *down*. Freivalds and Smith [FS93], as well as [ACJS04], employed in learning theory *notations for constructive ordinals* [Rog67, § 11.7] as devices for algorithmic counting down.

Theorems 4 and 5 in Section 3 provide strong justification for studying the herein ordinal countdown variants of Postdictive Completeness.

Intuitively *ordinals* are representations of well-orderings. 0 represents the empty ordering, 1 represents the ordering of 0 by itself, 2 the ordering $0 < 1$, 3 the ordering $0 < 1 < 2$, ... The ordinal ω represents the standard ordering of all of $\mathbb{N}$. $\omega + 1$ represents, for example, the ordering of $\mathbb{N}$ consisting of the positive integers in standard order *followed by* 0. The *successor ordinals* are those of the form $\alpha + 1$ which have a single element laid out after a copy of another ordinal α. $\omega + \omega$ can be thought of as two copies of ω laid end to end — much bigger than ω. $\omega \cdot 3$ represents three copies of ω laid end to end. By contrast, $3 \cdot \omega$ represents ω copies of 3 — which is just ω. We see, then, for ordinals, $+, \cdot$ are not commutative. $\omega \cdot \omega$ is ω copies of ω laid out end to end. We can iterate this and define exponentiation for ordinals. *Limit ordinals* are those, like ω, $\omega + \omega$, $\omega \cdot \omega$, and ω^ω, which are not 0 and are not successor ordinals. All of them are infinite. Importantly, the *constructive ordinals* are just those that have a program (called a *notation*) in some system which specifies how to build them (lay them out end to end, so to speak).[3] Informally, here, for example, is how to think of counting down from such a notation for $\omega + \omega$. One first computes some estimate for a natural number to count down from and begins counting down from it; then, later, one can revise *once* this estimate and subsequently count down some more from that. For counting down from a notation for $\omega + \omega + \omega = \omega \cdot 3$, one can revise the initial estimate *twice*. Since ordinal notations represent well-orders, they do *not* permit infinitely long countdowns, neither algorithmic (we do finite, algorithmic countdowns) nor non-algorithmic.

[SSV04] gives a further generalized notion of counting down. They consider certain partial orders with no *computable* infinitely descending chains. In the present paper we consider arbitrary and also computable graphs with no infinite, computable paths, and we algorithmically count "down" along their paths. Theorem 4.11, in Section 4.2 below, gives a nice example of linearly ordered, computable such graph which nonetheless has infinite paths (just not computable ones). We call our graphs in the present paper, *countdown graphs*.

We make use of countdown graphs for delaying the requirement of postdictive completeness (respectively, consistency) by requiring a learner to start an *independent* countdown for each datum $g(i)$ seen and to be postdictively complete (respectively, consistent) regarding $g(i)$ as soon as the countdown for $g(i)$ terminates.[4]

Section 2 will introduce the notation and concepts used in this paper.

In the prior literature we also see further variants of postdictive completeness and consistency not based on delay. For example, [CJSW04] surveys with references these variants. Roughly, below, when we attach $\mathcal{R}$ to the front of a name of a criterion requiring postdictive completeness or consistency, this means that the associated learnability must be witnessed by a (total) computable learner as opposed to just a partial computable learner (defined at least where it minimally needs to be); when we attach a $\mathcal{T}$ to the front of a name of a criterion requiring postdictive completeness (respectively, consistency), this means that the associated learnability must be witnessed by a (total) computable learner which is postdictively complete (respectively, consistent) on *all* input functions regardless of whether the learner actually learns them.

[2] Note that, for $n = 0$, the data seen is empty and the output hypothesis, p_0, is unconstrained.

[3] Technically, we count down from *notations* for constructive ordinals (instead of from the ordinals themselves) simply because notations, being finite (programs), in principle, fit inside computers; whereas, at least infinite ordinals do not.

[4] Below we refer to a vector of such individual counts as a *multicount*.

Sections 3 and 4 present our results. All of our results in Section 3 provide information about polynomial time learners. Furthermore, some of our results in Section 4.1 entail learnability with linear time learners. These time bounds are uniform bounds on how much time it takes the learner to conjecture each hypothesis in terms of the total size of the input data it can use for making this conjecture. Suppose for discussion p is a polynomial time bound. Pitt [Pit89] notes that Ex-learning allows unfair postponement tricks, i.e., a learner h can put off outputting significant conjectures based on data σ until it has seen a much larger sequence of data τ so that $p(|\tau|)$ is enough time for h to think about σ as long as it needs.[5] In this way the polytime restriction on each output does not, by itself, have the desirable effect of constraining the total learning time. Pitt [Pit89] then lays out some additional constraints toward avoiding "cheating" by such postponement tricks. He discusses in this regard what we are calling postdictive completeness. He also considers further constraints since he wants to forbid enumeration techniques [JORS99]. For our complexity-bounded results in Section 4.1 we get by with an extremely fair, restricted kind of *linear-time* learner, we call *transductive*. A transductive learner has access only to its current datum.

In Section 3 we see, from Theorems 3.5 and 3.6 and the proof of the first, that, for polytime learning, postdictive completeness (and delayed variants): 1. allows *some* but *not* all postponement tricks, *and* 2. there is a surprisingly tight boundary, for polytime learning, between what postponement is allowed and what is not. For *example*: 1. the set of polytime computable functions *is* polytime postdictively completely Ex-learnable (by a complexity-bounded enumeration technique) employing some postponement, *but* 2. the set of exptime computable functions, while polytime Ex-learnable with a little more postponement, is *not* polytime postdictively completely Ex-learnable! From Theorem 3.5, we see that, for w a notation for ω, the set of exptime functions *is* polytime Ex-learnable with w-*delayed* postdictive completeness. Theorems 3.5 and 3.6 also provide generalizations to further, small constructive limit ordinals.

Section 4.1 shows how the different variants of our criteria relate in learning power. Our main theorem in this section is Theorem 4.3. For *example*, it entails that there is a set of computable functions which is postdictively *consistently* learnable (with no delays) by a transductive, linear time learner *but* is *not* postdictively *completely* learnable with delays employing *any* countdown graph.

In Section 4.2, our main result, Theorem 4.14, entails (including with Corollaries) *complete characterizations* of learning power *in dependence on* associated (computable) *countdown graphs*. Corollary 4.17 extends the finite hierarchy given in [AZ07] into the constructive transfinite.

Many of our proofs use recursion theorems and are a bit combinatorially difficult.

2 Mathematical Preliminaries

Any unexplained complexity-theoretic notions are from [RC94]. All unexplained general computability-theoretic notions are from [Rog67].

Strings herein are finite and over the alphabet $\{0,1\}$. $\{0,1\}^*$ denotes the set of all such strings; ε denotes the empty string.

$\mathbb{N}$ denotes the set of natural numbers, {0,1,2,...}. We do not distinguish between natural numbers and their *dyadic* representation as strings.[6]

For each $w \in \{0,1\}^*$ and $n \in \mathbb{N}$, w^n denotes n copies of w concatenated end to end. For each string w, we define $\text{size}(w)$ to be the length of w. As we identify each natural number x with its dyadic representation, for all $n \in \mathbb{N}$, $\text{size}(n)$ denotes the length of the dyadic representation of n. For all strings w, we define $|w|$ to be $\max\{1, \text{size}(w)\}$. [7]

The symbols $\subseteq, \subset, \supseteq, \supset$ respectively denote the subset, proper subset, superset and proper superset relation between sets.

For sets A, B, we let $A \smallsetminus B := \{a \in A \mid a \notin B\}$, $\overline{A} := \mathbb{N} \smallsetminus A$.

We sometimes denote a function f of $n > 0$ arguments $x_1, \ldots, x_n$ in lambda notation (as in Lisp) as $\lambda x_1, \ldots, x_n . f(x_1, \ldots, x_n)$. For example, with $c \in \mathbb{N}$, $\lambda x . c$ is the constantly c function of one argument.

A function ψ is *partial computable* iff there is a Turing machine computing ψ. $\mathcal{R}$ and $\mathcal{P}$ denote the set of all (total) computable and partial computable functions $\mathbb{N} \to \mathbb{N}$, respectively. If ψ is not defined

[5] Pitt talks in this context of *delaying tricks*. We changed this terminology due to the clash with Akama and Zeugmann's terminology for *delayed* postdictive completeness.

[6] The *dyadic* representation of a natural number $x :=$ the x-th finite string over $\{0,1\}$ *in lexicographical order*, where the counting of strings starts with zero [RC94]. Hence, unlike with binary representation, lead zeros matter.

[7] This convention about $|\varepsilon| = 1$ helps with runtime considerations.

for some argument x, then we denote this fact by $\psi(x)\uparrow$ and we say that ψ on x *diverges*. The opposite is denoted by $\psi(x)\downarrow$ and we say that ψ on x converges.

We say that a partial function ψ *converges to* p iff $\forall^\infty x : \psi(x)\downarrow = p$.

[RC94, §3] describes an *efficiently* numerically named or coded[8] programming system for multi-tape Turing machines (TMs) which compute the partial computable functions $\mathbb{N} \to \mathbb{N}$. Herein we name this system φ. φ_p denotes the partial computable function computed by the TM-program with code number p in the φ-system, and Φ_p denotes the partial computable *runtime* function of the TM-program with code number p in the φ-system. In the present paper, we employ a number of complexity bound results from [RC94, §§ 3 & 4] regarding (φ, Φ). These results will be clearly referenced as we use them.

Let

- LinPrograms $:= \{e \mid e \in \mathbb{N} \wedge \exists c, d \forall n \in \mathbb{N} : \Phi_e(n) \leq c \cdot |n| + d)\}$;
- $\mathbf{LinF} := \{\varphi_e \mid e \in \text{LinPrograms}\}$;
- PolyPrograms $:= \{e \mid e \in \mathbb{N} \wedge \exists p$ polynomial $\forall n \in \mathbb{N} : \Phi_e(n) \leq p(|n|)\}$; and
- $\mathbf{PF} := \{\varphi_e \mid e \in \text{PolyPrograms}\}$.

For $g \in \mathbf{LinF}$ we say that g is *computable in linear time*, for $g \in \mathbf{PF}$ we say that g is *computable in polytime*, or also, *feasibly computable*.[9]

We fix the 1-1 and onto pairing function $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ from [RC94], which is based on dyadic bit-interleaving. Pairing and unpairing is computable in linear time. π_1 and π_2, respectively, denote the unpairing into the left and right component of a given coded pair, respectively.

For all $f, g \in \mathcal{R}$ we let $\langle f, g \rangle$ denote $\lambda i . \langle f(i), g(i) \rangle$.

Whenever we consider sequences of natural numbers as input to TMs, it is understood that the general coding function $\langle \cdot, \cdot \rangle$ is used to (left-associatively) code the tuples into appropriate TM-input.

A *finite sequence* is a mapping with a finite initial segment of $\mathbb{N}$ as domain. $\varnothing$ denotes the empty sequence (and, also, the empty set). The set of all finite sequences of natural numbers is denoted by $\mathbb{S}\text{eq}$. For each finite sequence σ, we will denote the first element, if any, of that sequence by $\sigma(0)$, the second, if any, with $\sigma(1)$ and so on. $\#\text{elets}(\sigma)$ denotes the length of a finite sequence σ, that is, the size of its domain.

We will consider *infinite sequences* s as functions with domain $\mathbb{N}$, and denote them at position $x \in \mathbb{N}$ by $s(x)$.

$\diamond$ denotes concatenation on sequences; the second argument may be an infinite sequence, the first may not. We use infix notation when we use $\diamond$.

From now on, by convention, f, g and h with or without decoration range over (partial) functions $\mathbb{N} \to \mathbb{N}$, x, y with or without decorations range over $\mathbb{N}$ and σ, τ with or without decorations range over finite sequences of natural numbers.

Following [LV97], we define for all $x \in \mathbb{N}$: $\overline{x} = 1^{\#\text{elets}(x)}0x$. Using this notation we can define a function $\langle \cdot \rangle_{\mathbb{S}\text{eq}}$ coding arbitrarily long finite sequences of natural numbers into $\mathbb{N}$ (represented dyadically) such that

$$\langle \sigma \rangle_{\mathbb{S}\text{eq}} := \overline{\sigma(0)} \ldots \overline{\sigma(\#\text{elets}(\sigma) - 1)}. \tag{1}$$

For example the finite sequence $(4, 7, 10)_{decimal} = (01, 000, 011)_{dyadic}$ is coded as 11 0 01 1110 000 1110 011 (but without the spaces, which were added for ease of reading).[10]

Note that, for all $\sigma, \tau : \langle \sigma \diamond \tau \rangle_{\mathbb{S}\text{eq}} = \langle \sigma \rangle_{\mathbb{S}\text{eq}} \langle \tau \rangle_{\mathbb{S}\text{eq}}$. Also note that, for all $x \in \mathbb{N}$, $\overline{x}$ is equal to the code of the sequence of length 1 containing only x, and, for all $n \in \mathbb{N}$, $\overline{x}^n$ is equal to the code of the sequence of length n, each element being x.

For any finite sequence σ such that $\#\text{elets}(\sigma) > 0$, we let $\text{last}(\sigma)$ be the last element of σ and σ^- be σ with its last element deleted.

Obviously, $\langle \cdot \rangle_{\mathbb{S}\text{eq}}$ is 1-1 [LV97]. The set of all sequences is decidable in linear time. The time to encode a sequence, that is, to compute $\lambda k, v_1, \ldots, v_k . \langle v_1, \ldots, v_k \rangle_{\mathbb{S}\text{eq}}$ is $\mathcal{O}(\lambda k, v_1, \ldots, v_k . \sum_{i=1}^{k} |v_i|)$. Therefore, the size of the codeword is also linear in the size of the elements: $\lambda k, v_1, \ldots, v_k . |\langle v_1, \ldots, v_k \rangle_{\mathbb{S}\text{eq}}|$ is $\mathcal{O}(\lambda k, v_1, \ldots, v_k . \sum_{i=1}^{k} |v_i|)$.[11]

[8] This numerical coding guarantees that many simple operations involving the coding run in linear time. This is by contrast with historically more typical codings featuring prime powers and corresponding at least exponential costs to do simple things.

[9] We are mostly not considering herein interesting polytime probabilistic or quantum computing variants of the deterministic feasibility case.

[10] 1100111100001110011 is of course the dyadic representation of some number $\in \mathbb{N}$.

[11] For these $\mathcal{O}$-formulas, $|\varepsilon| = 1$ helps.

Furthermore we have

$$\forall x : 1 \leq \text{size}(\bar{x}); \tag{2}$$

$$\forall \sigma : \#\text{elets}(\sigma) \leq |\langle\sigma\rangle_{\mathbb{S}\text{eq}}|; \tag{3}$$

$$\lambda\langle\sigma\rangle_{\mathbb{S}\text{eq}}.\#\text{elets}(\sigma) \in \mathbf{LinF}; \tag{4}$$

$$\lambda\langle\sigma\rangle_{\mathbb{S}\text{eq}}, i.\begin{cases}\sigma(i), & \text{if } i < \#\text{elets}(\sigma);\\ 0, & \text{otherwise,}\end{cases} \in \mathbf{LinF}. \tag{5}$$

Henceforth, we will many times identify a finite sequence σ with its code number $\langle\sigma\rangle_{\mathbb{S}\text{eq}}$. However, when we employ expressions such as $\sigma(x)$, $\sigma = f$ and $\sigma \subset f$, we consider σ as a *sequence*, not as a number.

For a partial function g and $i \in \mathbb{N}$, if $\forall j < i : g(j)\downarrow$, then $g[i]$ is defined to be the finite sequence $g(0), \ldots, g(i-1)$.

For every set of functions $\mathcal{S} \subseteq \mathcal{R}$ we define $[\mathcal{S}] = \{\sigma \mid \exists g \in \mathcal{S} : \sigma \subseteq g\}$.

By s-m-n, there is patch computable such that, for all σ, e,

$$\forall x : \varphi_{\text{patch}(\sigma,e)}(x) = \begin{cases}\sigma(x), & \text{if } x \in \text{dom}(\sigma);\\ \varphi_e(x), & \text{otherwise.}\end{cases} \tag{6}$$

By [RC94, Theorem 3.13], there is patch_0 such that such that,

$$\text{patch}_0 \in \mathbf{LinF}, \tag{7}$$

$$\forall\sigma\forall x : \varphi_{\text{patch}_0(\sigma)}(x) = \begin{cases}\sigma(x), & \text{if } x < \#\text{elets}(\sigma);\\ 0, & \text{otherwise,}\end{cases} \tag{8}$$

and

$$\forall\sigma : \text{patch}_0(\sigma) \in \text{LinPrograms}. \tag{9}$$

Some of our proofs will use Kleene's Recursion Theorem (**KRT**) [Rog67, page 214, problem 11-4], a variant of Roger's recursion theorem, representing a form of individual self-reference.

In one case we will use a stronger theorem then **KRT**, namely we use the *Operator Recursion Theorem* (**ORT**) [Cas74]. **ORT** is a form of infinitary self-reference. That is, **ORT** provides a means of forming an infinite computable sequence of programs $P(0), P(1), \ldots$ such that each program $P(i)$ knows all programs in the sequence and its own index i. The function P can also be assumed monotone increasing (hence, 1-1); this is referred to as *padded* **ORT**. For a thorough explanation of **ORT** see [Cas94]. **ORT** generalizes Kleene's Parametric Recursion Theorem (**PKRT**). **PKRT** provides a means of forming an infinite computable sequence of programs $P(0), P(1), \ldots$ such that each program $P(i)$ knows its own program and its own index i, but does not necessarily know the other programs in the sequence.

A *pre-order* is a pair $(A, \leq_A)$ such that $\leq_A$ is a transitive and reflexive binary relation on A.

2.1 Systems of Ordinal Notations

Church and Kleene introduced systems of ordinal notations. Our definition follows Rogers [Rog67, § 11.7].

A *system of ordinal notations* is a pair $(\mathcal{N}, \leq_{\mathcal{N}})$ and associated functions $k_{\mathcal{N}}, p_{\mathcal{N}}, q_{\mathcal{N}} \in \mathcal{P}$ and $\nu_{\mathcal{N}}$ mapping $\mathcal{N}$ into the set of all ordinals, such that

- $\mathcal{N} \subseteq \mathbb{N}$;
- $\forall u, v \in \mathcal{N} : u \leq_{\mathcal{N}} v \Leftrightarrow \nu_{\mathcal{N}}(u) \leq \nu_{\mathcal{N}}(v)$;
- For all $u \in \mathcal{N}$: $\nu_{\mathcal{N}}(u) = 0 \Rightarrow k_{\mathcal{N}}(u) = 0$, $\nu_{\mathcal{N}}(u)$ is successor ordinal $\Rightarrow k_{\mathcal{N}}(u) = 1$ and $\nu_{\mathcal{N}}(u)$ is limit ordinal $\Rightarrow k_{\mathcal{N}}(u) = 2$;
- For all $u \in \mathcal{N}$: $\nu_{\mathcal{N}}(u)$ is successor ordinal $\Rightarrow \nu_{\mathcal{N}}(p_{\mathcal{N}}(u)) = \nu_{\mathcal{N}}(u) + 1$;
- For all $u \in \mathcal{N}$: $\nu_{\mathcal{N}}(u)$ is limit ordinal $\Rightarrow \varphi_{q_{\mathcal{N}}(u)}$ is a monotonic increasing computable function such that $\nu_{\mathcal{N}} \circ \varphi_{q_{\mathcal{N}}(u)}$ converges to $\nu_{\mathcal{N}}(u)$.[12]

[12] N.B. Kleene's $(O, \leq_O)$ [Rog67] is technically not an example system of ordinal notations — since $\leq_O$ *on all of* O has incomparable elements.

$(\mathcal{N}, \leq_{\mathcal{N}})$ is called *computably related* iff $\leq_{\mathcal{N}}$ is computable.

An ordinal α is called *constructive* iff it receives a notation in some system of ordinal notations.

For countdown in polynomial time, as required for Section 3, we use *feasibly related feasible systems of ordinal notations* [CKP07].

A system of ordinal notations $\mathcal{N}$ is called *feasible* iff

- $k_{\mathcal{N}}$, $p_{\mathcal{N}}$ and $\lambda u, 0^n.\varphi_{q_{\mathcal{N}}(u)}(n)$ are computable in polytime;
- there are polytime computable functions $+_{\mathcal{N}}$ and $\cdot_{\mathcal{N}}$ such that for all $u, v \in \mathcal{N}$, $\nu_{\mathcal{N}}(u +_{\mathcal{N}} v) = \nu_{\mathcal{N}}(u) + \nu_{\mathcal{N}}(v)$ and $\nu_{\mathcal{N}}(u \cdot_{\mathcal{N}} v) = \nu_{\mathcal{N}}(u) \cdot \nu_{\mathcal{N}}(v)$; and
- there are polytime computable functions $\underline{\cdot}_{\mathcal{N}}$, $l_{\mathcal{N}}$ and $n_{\mathcal{N}}$ such that, $\forall n \in \mathbb{N} : \nu_{\mathcal{N}}(\underline{n}_{\mathcal{N}}) = n$, $l_{\mathcal{N}}(u)$ is a notation for a limit ordinal and $\forall u \in \mathcal{N} : \nu_{\mathcal{N}}(u) = \nu_{\mathcal{N}}(l_{\mathcal{N}}(u)) + n_{\mathcal{N}}(u)$. [13]

$\mathcal{N}$ is called *feasibly related* iff $\leq_{\mathcal{N}}$ is feasibly decidable.

Note that for any constructive ordinal α, there is a computably related system of ordinal notations which gives a notation to α [Rog67]; furthermore, there is also a feasibly related feasible system of ordinal notations giving a notation to α .

2.2 Computational Limit Learning

In this paper we consider *several* indexed families of learning criteria. We proceed somewhat abstractly to avoid needless terminological repetitions.

For each $\mathcal{C} \subseteq \mathcal{P}$ and $\delta \subseteq \mathcal{R}^2$, we say that the pair $(\mathcal{C}, \delta)$ is a *learning criterion* (for short, *criterion*). The set $\mathcal{C}$ is called a *learner admissibility restriction*, and intuitively serves as a limitation on what functions will be considered as learners. Typical learner admissibility restrictions are $\mathcal{P}, \mathcal{R}$, as well as complexity classes. The predicate δ is called a *sequence acceptance criterion*, intuitively restricting what output-sequences by the learner are considered a successful learning of a given function. For $h \in \mathcal{P}, g \in \mathcal{R}$ we say that h $(\mathcal{C}, \delta)$-learns g iff $h \in \mathcal{C}$ and $(\lambda x.h(g[x]), g) \in \delta$. For $h \in \mathcal{P}, g \in \mathcal{R}$, we call $\lambda x.h(g[x])$ the *learning-sequence* of h given g. Here's an *example* δ, herein called **Ex**. Let $\mathbf{Ex} = \{(\langle p, d\rangle, q) \in \mathcal{R}^2 \mid p \text{ converges to some } e \ \wedge \ \varphi_e = q\}$. Intuitively, $(\langle p, d\rangle, q) \in \mathbf{Ex}$ means that the learning-sequence $\langle p, d\rangle$ successfully learns the function q iff: for some i, $p(i)$ is a correct program number for q, and this hypothesized program number will never change after that point i. N.B. For *this* example, the learning-sequence is a sequence of coded pairs and **Ex** completely disregards the second component d. Some *other* sequence acceptance criteria below make use of d as an auxiliary output of the learner. In these cases, d will code countdowns until some events of interest must happen. For $h \in \mathcal{P}$ and $\mathcal{S} \subseteq \mathcal{R}$ we say that h $(\mathcal{C}, \delta)$-learns $\mathcal{S}$ iff, for all $g \in \mathcal{S}$, h $(\mathcal{C}, \delta)$-learns g. The set of $(\mathcal{C}, \delta)$-learnable sets of computable functions is $\mathcal{C}\delta := \{\mathcal{S} \subseteq \mathcal{R} \mid \exists h \in \mathcal{C} : h \ (\mathcal{C}, \delta)\text{-learns } \mathcal{S}\}$. Instead of writing the pair $(\mathcal{C}, \delta)$, we will ambiguously write $\mathcal{C}\delta$. We will omit $\mathcal{C}$ if $\mathcal{C} = \mathcal{P}$.[14] One way to *combine* two sequence acceptance criteria δ and δ' is to intersect them as sets. We write $\delta\delta'$ for the intersection, and we present examples featuring countdowns in the next section.

We can turn a given sequence acceptance criterion δ into a learner admissibility restriction $\mathcal{T}\delta$ by admitting only those learners that obey δ *on all input*: $\mathcal{T}\delta := \{h \in \mathcal{P} \mid \forall g \in \mathcal{R} : (\lambda x.h(g[x]), g) \in \delta\}$.

2.3 Dynamically Bounded Postdiction

The following two definitions formalize the intuitive discussion about countdown graphs as given above in Section 1.

Definition 2.1. *A* graph *is a pair $(G, \rightarrow)$, where $G \subseteq \mathbb{N}$ and $\rightarrow$ is a binary relation on G. We will use infix notation for $\rightarrow$. For each graph $(G, \rightarrow)$, we say that τ is a* G-path *iff* $\#\text{elets}(\tau) > 0$, $\forall i < \#\text{elets}(\tau) : \tau(i) \in G$ *and* $\forall i < \#\text{elets}(\tau) - 1 : \tau(i) \rightarrow \tau(i+1)$. *For each graph G, let $\vec{G}$ denote the set of all G-paths. (S, R) is a* subgraph *of $(G, \rightarrow)$, iff $S \subseteq G$ and $R =\!\rightarrow |(S \times S)$. For all $m, n \in \mathbb{N}$, we write $m \rightarrow^* n$ (respectively, $m \rightarrow^+ n$) iff there is a G-path τ such that $\tau(0) = m$,* $\text{last}(\tau) = n$ *(respectively, additionally*

[13] The tally argument, 0^n, in the first bullet just above, is used in place of n to provide sufficient computational complexity resource for any uses of $q_{\mathcal{N}}$. N.B. In Section 3 of the present paper, we do not need to use the $q_{\mathcal{N}}$s; we get by with employing the feasibility of some of the other feasible functions: $+_{\mathcal{N}}$, $\cdot_{\mathcal{N}}$, ...

[14] Thus, every sequence acceptance criterion denotes at the same time a learning criterion and the set of learnable sets. It will be clear from context which meaning is intended. An example: **Ex**, then, denotes sequence acceptance criterion **Ex**, learning criterion $(\mathcal{P}, \mathbf{Ex})$ and set $\mathcal{P}\mathbf{Ex}$ of $(\mathcal{P}, \mathbf{Ex})$-learnable sets.

$\#\text{elets}(\tau) > 1$). *We sometimes write* G *for* $(G, \to)$. *A graph* $(G, \to)$ *is said to be* computable *iff* G *and* $\to$ *are computable. Note that a graph* G *is computable iff* $\tilde{G}$ *is computable. For a graph* $(G, \to)$ *we sometimes identify* $m \in G$ *with* $\{n \in G \mid m \to^+ n\}$. *With every pre-order* $(A, \leq_A)$ *we associate the graph* $(A, >_A)$, *where, for all* $a, b \in A$, $a >_A b$ *iff* $(b \leq_A a$ *and* $a \not\leq_A b)$.

Definition 2.2. *A graph* $(G, \to)$ *is called a* countdown graph, *iff* $\neg\exists r \in \mathcal{R} \forall i \in \mathbb{N} : r(i) \to r(i+1)$. *Note that if* G *is a countdown graph, then so is every subgraph of* G. *Let* $\mathcal{G}$ *and* $\mathcal{G}_{\text{comp}}$, *respectively, denote the set of all and all computable countdown-graphs, respectively.*

Example countdown graphs can be obtained from systems of ordinal notations. Let $(\mathcal{N}, \leq_{\mathcal{N}})$ be a system of ordinal notations. Then, $(\mathcal{N}, \leq_{\mathcal{N}})$ is a pre-order without infinite descending chains, so the graph associated with $(\mathcal{N}, \leq_{\mathcal{N}})$ is a countdown graph. If $(\mathcal{N}, \leq_{\mathcal{N}})$ is computably related, then the associated graph will be computable.

In Theorem 4.11 below we give one example of a countdown graph not based on a system of ordinal notations. Section 4.2 shows the impact of using these different kinds of countdown graphs for our purposes described below.

Soon we define what postdictive consistency, respectively completeness, with respect to $G \in \mathcal{G}$ means. Intuitively, every learner is required to have two outputs: a hypothesis, and a countdown output. For any learnee $g \in \mathcal{R}$, if the learner sees $g[i]$, the countdown output will need to encode one countdown for each $j < i$. As soon as the countdown for a given data item is over, the hypothesis has to be postdictively consistent, respectively complete, *for that data item.* We will refer to the countdown output of a learner as a *multicount* (as it represents more than one countdown). We refer to an learning-output of hypothesis and multicount as a hypothesis-multicount.

Definition 2.3. *The set of all* multicountdown sequences *is defined as*

$$\mathbb{M} := \{\sigma \in \mathbb{S}\text{eq} \mid \forall i < \#\text{elets}(\sigma) : (\sigma(i) \in \mathbb{S}\text{eq} \wedge \#\text{elets}(\sigma(i)) = i)\}.^{15}$$

An example multicountdown sequence is $\sigma_0 := \langle\rangle_{\mathbb{S}\text{eq}}$, $\langle 3\rangle_{\mathbb{S}\text{eq}}$, $\langle 2, 3\rangle_{\mathbb{S}\text{eq}}$, $\langle 1, 2, 2\rangle_{\mathbb{S}\text{eq}}$, $\langle 0, 1, 2, 1\rangle_{\mathbb{S}\text{eq}}$, $\langle 0, 0, 2, 2, 2\rangle_{\mathbb{S}\text{eq}}$, $\langle 2, 0, 2, 3, 1, 1\rangle_{\mathbb{S}\text{eq}}$, $\langle 0, 0, 2, 7, 0, 0, 5\rangle_{\mathbb{S}\text{eq}}$.
σ_0 can be displayed as a matrix like this:

$$\begin{array}{c|c} n \backslash x & \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \hline \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} & \left(\begin{array}{cccccccc} 3 & 2 & 1 & 0 & 0 & 2 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 3 & 7 \\ 2 & 1 & 0 \\ 1 & 0 \\ 5 \end{array}\right) \end{array}. \tag{10}$$

In (10) each column is a multicount. For example, column $x = 4$ represents the multicount $\sigma_0(4) = \langle 0, 1, 2, 1\rangle$. Each row of (10) provides the successive values of a particular countdown. For example, each row of (10) (without initial empty entries) is the n-th countdown of σ_0. As we will see below, for an associated learnee g, the n-th row will be relevant to $g(n)$.

Definition 2.4. *For each* $\sigma \in \mathbb{M}$ *and* $n < \#\text{elets}(\sigma) - 1$ *we define*

$$\text{row}(n, \sigma) := \langle \sigma(n+1)(n), \ldots, \sigma(\#\text{elets}(\sigma) - 1)(n)\rangle_{\mathbb{S}\text{eq}}. \tag{11}$$

For σ_0 as presented above in (10), we have, for example $\text{row}(4, \sigma_0) = \langle 2, 1, 0\rangle_{\mathbb{S}\text{eq}}$. Each $\text{row}(n, \sigma)$ is a countdown.

We will consider a given countdown sequence τ as *terminated* with respect to a given countdown graph $G \in \mathcal{G}$, iff $\tau \notin \tilde{G}$. We then say that "τ has terminated" or "τ has bottomed out". For a given multicountdown sequence we will define the set of all n such that the n-th countdown has (started and) bottomed out just below.

[15] Of course, $\sigma(i) \in \mathbb{S}\text{eq}$ means that the number $\sigma(i)$ is the code of a sequence.

Definition 2.5. *For all σ and all $G \in \mathcal{G}$, define*

$$\bot_G(\sigma) = \{n < \#\text{elets}(\sigma) \mid \sigma \notin \mathbb{M} \vee \text{row}(n,\sigma) \notin \vec{G}\}. \tag{12}$$

Furthermore, define $\perp\!\!\!\perp_G(\varnothing) = \bot_G(\varnothing)$ and, for $\sigma \neq \varnothing$, $\perp\!\!\!\perp_G(\sigma) = \bot_G(\sigma) \setminus \bot_G(\sigma^-)$.[16] *We omit the subscript G whenever no confusion can arise.*

We pronounce $\bot$ as "bottom" and $\perp\!\!\!\perp$ as "recent bottom". For $\sigma \in \mathbb{M}$, $\bot(\sigma)$ is the set of all countdown numbers where the countdown has terminated, while $\perp\!\!\!\perp(\sigma)$ is the set of countdowns that have intuitively "just now" terminated.

Let us, for example, consider the finite countdown graph G on $\{0,1,2,3\}$ with the natural $>$-order on $\mathbb{N}$. For σ_0 depicted above in (10), we have $\bot_G(\sigma_0) = \{0,1,2,3,6\}$. The example of rows $n = 4$ and $n = 5$ shows that reaching a minimal element (in this case 0) of G does not imply immediate termination of the countdown. The example of rows $n = 2$ and $n = 3$ shows how countdowns terminate when not obeying the graph relation. Note that the countdown for row $n = 6$ has terminated immediately when it started, as it started with 5, and $\langle 5\rangle_{\mathbb{S}\text{eq}}$ is not a G-path. From rows $n = 4$ and $n = 6$ we see that the different countdowns do not have to terminate in row order.

Next we define two families of sequence acceptance criteria, employing countdowns as described above. The rest of the paper will be concerned with studying these criteria in various settings.

Definition 2.6. *For $G \in \mathcal{G}$ let, for all $p, d, q \in \mathcal{R}$,*

- $\mathbf{Pcs}_G(\langle p,d\rangle, q) :\Leftrightarrow \forall x \forall n \in \bot_G(d[x]) : \varphi_{p(x)}(n)\!\downarrow \Rightarrow \varphi_{p(x)}(n) = q(n)$ *and*
- $\mathbf{Pcp}_G(\langle p,d\rangle, q) :\Leftrightarrow \forall x \forall n \in \bot_G(d[x]) : \varphi_{p(x)}(n)\!\downarrow = q(n)$.

For all $g \in \mathcal{R}$ and $h, f \in \mathcal{P}$, we say that $\langle h, f\rangle$ works postdictively consistently (respectively, completely) on g with G-delay *iff $(\langle h, f\rangle, g) = (\lambda i.(\langle h(g[i]), f(g[i])\rangle), g) \in \mathbf{Pcs}_G$ (respectively, $\mathbf{Pcp}_G$). We omit "with G-delay", if no confusion can arise.*

For notational purposes, we define the following variants on row, $\bot$ and $\perp\!\!\!\perp$.

Definition 2.7. *For all $G \in \mathcal{G}$, $\sigma \in \mathbb{S}\text{eq}$, $f \in \mathcal{P}$ and $n \leq \#\text{elets}(\sigma)$, define*

$$\text{row}(n, f, \sigma) := \text{row}(n, \lambda i \leq \#\text{elets}(\sigma).f(\sigma[i])), \tag{13}$$

$$\bot_G(f, \sigma) := \bot_G(\lambda i \leq \#\text{elets}(\sigma).f(\sigma[i])), \tag{14}$$

$$\perp\!\!\!\perp_G(f, \sigma) :=\perp\!\!\!\perp_G(\lambda i \leq \#\text{elets}(\sigma).f(\sigma[i])). \tag{15}$$

We omit the subscript G whenever no confusion can arise.

Note that, for all $f \in \mathcal{P}$, all σ and all $n < \#\text{elets}(\sigma)$,

$$\text{row}(n, f, \sigma) = \lambda i \leq \#\text{elets}(\sigma) - n - 1.f(\sigma[i + n + 1])(n). \tag{16}$$

Also we have, for all $G \in \mathcal{G}$ and all $f \in \mathcal{P}$ such that $\forall \tau : (\lambda i \leq \#\text{elets}(\tau).f(\tau[i])) \in \mathbb{M}$, for all $\sigma \in \mathbb{S}\text{eq}$ and all $n < \#\text{elets}(\sigma)$,

$$n \in \bot_G(f, \sigma) \Leftrightarrow \text{row}(n, f, \sigma) \notin \vec{G}. \tag{17}$$

3 Complexity Results

For this section only, let $\mathcal{N}$ be a feasibly related feasible system of ordinal notations for at least the ordinals $< \omega^2$. Let w be a notation for ω in $\mathcal{N}$. For each $n \in \mathbb{N}$, $\underline{n}$ denotes a notation for n in $\mathcal{N}$, such that $\lambda n.\underline{n}$ is computable in polytime. We will assume for all constructive ordinals α,

$$\forall n \in \mathbb{N}, u \in \mathcal{N} : (u \text{ is notation in } \mathcal{N} \text{ for } \alpha + n) \Rightarrow n \leq u.^{17} \tag{18}$$

[16] Note that $\perp\!\!\!\perp_G(\varnothing) = \bot_G(\varnothing) = \varnothing$.

[17] Specific systems of ordinal notations seen in the literature typically, perhaps always, satisfy (18).

Definition 3.1. *Let* exp *denote the function* $\lambda x.2^x$. *Furthermore, for all* n, *we write* $\exp^n$ *for the* n*-times application of* exp. *In particular,* $\exp^0$ *denotes the identity. For all* k *let*

$$\begin{aligned} \mathrm{Exp}_k\mathrm{Programs} &:= \{e \mid e \in \mathbb{N} \wedge \exists p \text{ polynomial } \forall n \in \mathbb{N} : \Phi_e(n) \leq exp^k(p(|n|))\}; \\ \mathbf{EXP}_k\mathbf{F} &:= \{\varphi_e \mid e \in \mathrm{Exp}_k\mathrm{Programs}\}; \\ \mathrm{ExpPrograms} &:= \mathrm{Exp}_1\mathrm{Programs} \\ \mathbf{EXPF} &:= \mathbf{EXP}_1\mathbf{F} \end{aligned}$$

For $g \in \mathbf{PF}$ we say that g is *computable in polytime*, or also, *feasibly computable*. Recall that we have, by (3), $\forall \sigma : \#\mathrm{elets}(\sigma) \leq |\sigma|$.

Definition 3.2. *Let* S, T *be such that*

$$\forall p, x, t : S(p,x,t) = \begin{cases} \varphi_p(x), & \text{if } \Phi_p(x) \leq |t|; \\ 0, & \text{otherwise;} \end{cases} \tag{19}$$

$$\forall p, x, t : T(p,x,t) = \begin{cases} 1, & \text{if } \Phi_p(x) \leq |t|; \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

Lemma 3.3. *There exist linear time computable functions* min *and* all, *such that*

$$\forall p, x, m : \varphi_{\min(p)}(x,m) = \begin{cases} y, & \text{if } y \text{ is the least number} \leq |m| \text{ such that:} \\ & \varphi_p(x,y)\!\downarrow \neq 0 \wedge \forall z < y : \varphi_p(x,z) = 0; \\ |m|+1, & \text{if } \forall z \leq |m| : \varphi_p(x,z)\!\downarrow = 0; \\ \uparrow, & \text{otherwise;} \end{cases} \tag{21}$$

$$\forall p, x, m : \varphi_{\mathrm{all}(p)}(x,m) = \begin{cases} 1, & \text{if } \forall z \leq \#\mathrm{elets}(m) : \varphi_p(x,z)\!\downarrow \neq 0; \\ 0, & \text{if } \forall z \leq \#\mathrm{elets}(m) : \varphi_p(x,z)\!\downarrow \wedge \\ & \quad \exists z \leq \#\mathrm{elets}(m) : \varphi_p(x,z)\!\downarrow = 0; \\ \uparrow, & \text{otherwise,} \end{cases} \tag{22}$$

and, for all $p \in \mathrm{PolyPrograms}$, $\min(p), \mathrm{all}(p) \in \mathrm{PolyPrograms}$.

Proof. This follows from [RC94, Lemmas 3.15 & 3.16] and (3). □

Lemma 3.4. *Regarding timebounded computability, we have the following.*

- #elets *is computable in polytime by (4).*
- *Equality checks,* $\lfloor \log \rfloor$ *and* $\dot{-}$ *are computable in linear time [RC94, Lemma 3.2].*
- $\lambda k.2^k$ *is computable in time* $\mathcal{O}(k)$ *[RC94, Lemma 3.2].*
- *Conditional definition is computable in a time polynomial in the runtimes of its defining programs [RC94, Lemma 3.14].*
- *Boolean combinations of predicates computable in polytime are computable in polytime [RC94, Lemma 3.18].*
- *From [RC94, Corollary 3.7]* S *and* T *from (19) and (20) above are polytime computable.*[18]
- *By (7),* patch_0 *is computable in polytime.*

Theorem 3.5. *(a)* $\mathbf{PF} \in \mathbf{PFPcp}_{\underline{0}}\mathbf{Ex}$.

(b) $\mathbf{EXPF} \in \mathbf{PFPcp}_w\mathbf{Ex}$.

(c) $\forall n : \mathbf{EXP}_n\mathbf{F} \in \mathbf{PFPcp}_{w \cdot \underline{n}}\mathbf{Ex}$.

Furthermore, each of (a), (b) and (c) is witnessed by a respective learner $\langle h, f \rangle$ *such that* $\mathrm{range}(h) \subseteq$ PolyPrograms, $\subseteq$ ExpPrograms *and* $\subseteq \mathrm{Exp}_n$Programs, *respectively.*

[18] N.B. S and T above are variants of the S and T featured in [RC94, Corollary 3.7].

Note that (a) and (b) are *both* special cases of (c). We will prove (a) in detail and will then give a sketch as to how this proof can be generalized to a proof of (c).

Proof of (a). This proof employs a complexity-bounded enumeration technique [JORS99].

By [RC94, Theorems 4.13(b) & 4.17] there is a linear time computable e such that $\mathbf{PF} = \{\varphi_{e(j)} \mid j \in \mathbb{N}\}$ and $\forall j \in \mathbb{N} : e(j) \in \text{PolyPrograms}$. Then, by Lemma 3.4, it is easy to see that there is $h \in \mathbf{PF}$ such that[19]

$$\forall \sigma : h(\sigma) = \begin{cases} e(j), & \text{if there is a minimal } j \leq |\sigma| : \forall x < \#\text{elets}(\sigma) : \\ & (T(e(j), x, \sigma) \wedge\ S(e(j), x, \sigma) = \sigma(x)); \\ \text{patch}_0(\sigma), & \text{otherwise.} \end{cases} \tag{23}$$

To show that h converges on all $g \in \mathbf{PF}$: Let $g \in \mathbf{PF}$. Let j_0 be minimal such that $\varphi_{e(j_0)} = g$. Let p be a polynomial such that $\forall x : \Phi_{e(j_0)}(x) \leq p(|x|)$. We then have the following.

- $\forall^\infty n, j_0 \leq n \leq |g[n]|$ (by (3)).
- $\forall^\infty n \forall j < j_0 : g[n] \not\subseteq \varphi_{e(j)}$ (as j_0 minimal such that $\varphi_{e(j_0)} = g$).
- We have $\forall^\infty x : \Phi_{e(j_0)}(x) \leq x$.[20] Hence, $\forall^\infty n \forall x \leq n : \Phi_{e(j_0)}(x) \leq n$.[21] Therefore, using (3), $\forall^\infty n \forall x < n : T(e(j_0), x, g[n])$; hence, also $\forall^\infty n \forall x < n : S(e(j_0), x, g[n]) = \varphi_{e(j_0)}(x) = g(x)$.

By the three items above, we have $\forall^\infty n : h(g[n]) = e(j_0)$. Let $f = \lambda \sigma . \underline{0}$. Obviously, $\langle h, f \rangle$ witnesses $\mathbf{PF} \in \mathbf{Pcp}_{\underline{0}}\mathbf{Ex}$. The furthermore clause follows from the choice of e and patch_0. □ (FOR (a))

Proofsketch of (c). Define

$$\forall \sigma, \forall k < \#\text{elets}(\sigma) : f_k(\sigma) = \begin{cases} w \cdot \underline{n-1} + \underline{\exp^1(k) \dot{-} \#\text{elets}(\sigma)}, & \text{if } \#\text{elets}(\sigma) \leq \exp^1(k); \\ \vdots & \vdots \\ w \cdot \underline{1} + \underline{\exp^{n-1}(k) \dot{-} \#\text{elets}(\sigma)}, & \text{else if } \#\text{elets}(\sigma) \leq \exp^{n-1}(k); \\ w \cdot \underline{0} + \underline{\exp^{n}(k) \dot{-} \#\text{elets}(\sigma)}, & \text{otherwise.} \end{cases} \tag{24}$$

We define $f \in \mathbf{PF}$ by

$$\forall \sigma : f(\sigma) = \langle f_0(\sigma), \ldots, f_{\#\text{elets}(\sigma)-1}(\sigma) \rangle_{\mathbb{S}\text{eq}}.$$

Theorem 3.6. *(a)* $\forall n \in \mathbb{N} : \mathbf{EXPF} \not\subseteq \mathbf{PFPcp}_{\underline{n}}\mathbf{Ex}$.

(b) $\forall k, n \in \mathbb{N} : \mathbf{EXP}_{k+1}\mathbf{F} \not\subseteq \mathbf{PFPcp}_{w \cdot k + \underline{n}}\mathbf{Ex}$.

Proof of (a). Suppose by way of contradiction otherwise as witnessed by n and $\langle h, f \rangle$. Note that $[\mathbf{EXPF}] = \mathbb{S}\text{eq}$; thus, $\langle h, f \rangle \in \mathcal{T}\mathbf{Pcp}_{\underline{n}}$ (see Remark 4.1 below).

Define $g \in \mathcal{R}$ according to the following informal definition in stages. g_s denotes g as defined until before stage s.

$g_0 = \varepsilon$
stage $s = 0$ to ∞
  if $h(g_s \diamond \overline{0} \diamond \overline{0}^n) = h(g_s)$
    then $g_{s+1} = g_s \diamond \overline{1} \diamond \overline{0}^n$
    else $g_{s+1} = g_s \diamond \overline{0} \diamond \overline{0}^n$

Claim 1: h does not converge on g.

We show the claim by showing $\forall s : h(g_{s+1}) \neq h(g_s)$. As $\langle h, f \rangle \in \mathcal{T}\mathbf{Pcp}_{\underline{n}}$, we have for all $s \in \mathbb{N}$ and each $j \in \{0, 1\}$, $\lambda i \leq n . f(g_s \diamond \overline{j} \diamond \overline{0}^i)$ is not a $\underline{n}$-path, as there is no path of length $n+1$ in $\underline{n}$; hence, $\varphi_{h(g_s \diamond \overline{j} \diamond \overline{0}^n)}(\#\text{elets}(g_s)) = j$.

If now $h(g_s \diamond \overline{0} \diamond \overline{0}^n) = h(g_s)$, then $\varphi_{h(g_{s+1})}(\#\text{elets}(g_s)) = \varphi_{h(g_s \diamond 1 \diamond \overline{0}^n)}(\#\text{elets}(g_s)) = 1 \neq \overline{0} = \varphi_{h(g_s \diamond \overline{0} \diamond \overline{0}^n)}(\#\text{elets}(g_s)) = \varphi_{h(g_s)}(\#\text{elets}(g_s))$; thus, $h(g_{s+1}) \neq h(g_s)$.

If $h(g_s \diamond \overline{0} \diamond \overline{0}^n) \neq h(g_s)$, then $h(g_{s+1}) = h(g_s \diamond \overline{0} \diamond \overline{0}^n) \neq h(g_s)$. □ (FOR CLAIM 1)

[19] Recall that the properties of patch_0 are listed in (7-9).

[20] By [RC94, §2.5, (9)], there are $a, b \in \mathbb{N}$ such that $\forall x : 2^{|x|} \leq a \cdot x + b$; thus, there is $d > 0$ such that $\forall^\infty x : 2^{|x|} \leq d \cdot x$. Clearly, $\forall^\infty x : p(|x|) \leq \frac{1}{d} 2^{|x|}$. Thus, $\forall^\infty x : p(|x|) \leq x$.

[21] Let n_0, n_1 be such that $\forall x \geq n_0 : \Phi_{e(j_0)}(x) \leq x$ and $\forall x < n_0 : \Phi_{e(j_0)}(x) \leq n_1$. Then, for all $n \geq \max\{n_0, n_1\}$ and for all $x \leq n$, we have (if $x < n_0$) $\Phi_{e(j_0)}(x) \leq n_1 \leq n$, and (otherwise) $\Phi_{e(j_0)}(x) \leq x \leq n$.

Claim 2: $g \in \mathbf{EXPF}$.
By the construction of g, we have $\forall s : g_s \in \{0,1\}^{s\cdot(n+1)}$. Hence, to compute $g(x)$ for any given x, it suffices to execute stages 0 through x of the above algorithm to get g_{x+1}, from which $g(x)$ can then be extracted. Therefore, it suffices to show that, for all s, the stages 0 through s of the above algorithm can be done with an appropriate timebound.

Let p be a polynomial upper-bounding the runtime of h such that $\forall x : x \leq p(x)$. For any stage s, the time to execute stage s is in $\mathcal{O}(\lambda s.p(|g_s \diamond \overline{0}^{n+1}|) + p(|g_s|)) = \mathcal{O}(\lambda s.p(|g_s| + n + 1)) =^{22} \mathcal{O}(\lambda s.p(s\cdot(n+1)) + n + 1)) = \mathcal{O}(\lambda s.p(s))$. Therefore, for all s, the time to execute all stages 0 to s is bounded above by $\mathcal{O}(\lambda s.(s+1)\cdot p(s)) \subseteq \mathcal{O}(\lambda s.2^{p'(|s|)})$ for some polynomial p'.[23] □ (FOR CLAIM 2) □ (FOR (a))

Proof of (b). Suppose by way of contradiction otherwise as witnessed by $\langle h, f\rangle \in \mathbf{PF}$. The proof requires a different definition of g as follows.

$g_0 = \varepsilon$
`stage` $s = 0$ `to` ∞
  `loop until` $\#\text{elets}(g_s) \in \bot(f, g_s \diamond \overline{0} \diamond \overline{0}^i)$
    $i := i + 1$
  `loop until` $\#\text{elets}(g_s) \in \bot(f, g_s \diamond \overline{1} \diamond \overline{0}^j)$
    $j := j + 1$
  `if` $h(g_s \diamond \overline{0} \diamond \overline{0}^i) = h(g_s)$
    `then` $g_{s+1} = g_s \diamond \overline{1} \diamond \overline{0}^j$;
    `else` $g_{s+1} = g_s \diamond \overline{0} \diamond \overline{0}^i$.

Let p be a polynomial bounding the runtime of h and f, as well as deciders for S and $<_S$. Let s be a stage, set $x := \#\text{elets}(g_s)$.
Claim: There is a polynomial p' such that each loop will terminate after at most $exp^k(|p'(x)|)$ steps.
Proof. Let $m \in \{0,1\}$. Clearly, $f_x(\sigma \diamond \overline{m} \diamond \overline{0}^n) <_S w \cdot k$. By runtime considerations and (18) we see $f_x(g_s \diamond \overline{m} \diamond \overline{0}^n) <_S w\cdot(k-1) + \underline{exp(p(x_s + n + 1))}$; hence, for some polynomial p_1, $f_x(\sigma \diamond \overline{m} \diamond \overline{0}^{exp(p_1(x_s))}) <_S f_x(\sigma \diamond \overline{m} \diamond \overline{0}^{n+exp(p(x_s+n+1))}) <_S w\cdot k - 1$. Inductively one can now see that there is a polynomial p_{k-1} such that

$$f_x(\sigma \diamond \overline{m} \diamond \overline{0}^{exp^{k-1}(p_{k-1}(x_s))}) <_S w; \tag{25}$$

in particular, one can see that there is a polynomial p_k such that $x \in \bot(f, 0 \diamond \overline{m} \diamond \overline{0}^{exp^k(p_k(x_s))})$.
□ (FOR CLAIM)

Using [RC94, Theorem 3.17], one can now see $g \in \mathbf{EXP}_{k+1}\mathbf{F}$. The rest of the proof is analogous to the proof of (a). □ (FOR (b))

4 General Results

Remark 4.1. *Obviously, we have for all $G \in \mathcal{G}$ and $\mathcal{S} \subseteq \mathcal{R}$ such that $[\mathcal{S}] = \mathbb{S}\text{eq}$*

$$\mathcal{S} \in \mathcal{R}\mathbf{Pcs}_G\mathbf{Ex} \cup \mathbf{Pcs}_G\mathbf{Ex} \Rightarrow \mathcal{S} \in \mathcal{T}\mathbf{Pcs}_G\mathbf{Ex}; \tag{26}$$
$$\mathcal{S} \in \mathcal{R}\mathbf{Pcp}_G\mathbf{Ex} \cup \mathbf{Pcp}_G\mathbf{Ex} \Rightarrow \mathcal{S} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}. \tag{27}$$

The lemma just below encapsulates diagonalizations we employ in several proofs in the present section.

Lemma 4.2. *Let $\mathcal{C} \subseteq \mathcal{P}$. Let $\delta \in \mathcal{O}_2$ such that $\delta \subseteq \mathbf{Ex}$. Let $\mathcal{S} \subseteq \mathcal{R}$ such that $(\forall \langle h, f\rangle \in \mathcal{C} \mid \langle h, f\rangle$ δ-learns $\mathcal{S})\exists t_0, t_1 \in \mathcal{P}\forall e \in \mathbb{N} : \exists \mathcal{S}_e \subseteq \mathcal{R}$ such that (i) and (ii) just below hold.*

(i) $([\varphi_e] \subseteq [\mathcal{S}_e] \wedge \varphi_e \in \mathcal{R}) \Rightarrow \varphi_e \in \mathcal{S}$; *and*

[22] $\mathcal{O}(|g_s|) = \mathcal{O}(\#\text{elets}(g_s))$.
[23] Find k such that $\mathcal{O}(p) = \mathcal{O}(\lambda x.x^k)$. By [RC94, §2.5, (9)], there are a, b such that $x \leq a\cdot 2^{|x|} + b$. Thus, there is are c, d, c', d' such that $\forall x : p(x) \leq c\cdot x^k + d \leq c\cdot(a\cdot 2^{|x|} + b)^k + d \leq c'\cdot 2^{k\cdot|x|} + d'$.

(ii) For all $\sigma \in [\mathcal{S}_e]$, (iia)-(iie) just below hold.

(a) $\varphi_e = \sigma \Rightarrow$
$(t_0(e,\sigma)\downarrow \wedge\ t_1(e,\sigma)\downarrow \wedge\ h(\sigma \diamond t_0(e,\sigma))\downarrow \wedge h(\sigma)\downarrow)$

(b) $(t_0(e,\sigma)\downarrow \wedge t_1(e,\sigma)\downarrow) \Rightarrow\ (t_0(e,\sigma), t_1(e,\sigma) \in \mathbb{S}\mathrm{eq} \wedge \#\mathrm{elets}(t_0(e,\sigma)), \#\mathrm{elets}(t_1(e,\sigma)) > 0)$.

(c) $\sigma \subseteq \varphi_e \wedge h(\sigma \diamond t_0(e,\sigma))\downarrow \neq h(\sigma)\downarrow \Rightarrow \sigma \diamond t_0(e,\sigma) \in [\mathcal{S}_e]$.

(d) $\sigma \subseteq \varphi_e \wedge h(\sigma \diamond t_0(e,\sigma))\downarrow = h(\sigma)\ \wedge\ t_1(e,\sigma)\downarrow \Rightarrow \sigma \diamond t_1(e,\sigma) \in [\mathcal{S}_e]$.

(e) $(t_0(e,\sigma)\downarrow \wedge\ t_1(e,\sigma)\downarrow \wedge\ h(\sigma \diamond t_0(e,\sigma))\downarrow = h(\sigma) = h(\sigma \diamond t_1(e,\sigma))) \Rightarrow \varphi_{h(\sigma \diamond t_1(e,\sigma))} \notin \mathcal{R}$.

Then $\mathcal{S} \notin \mathcal{C}\delta$.

Proof. Suppose, by way of contradiction otherwise. Suppose $h \in \mathcal{C}$ witnesses $\mathcal{S} \in \mathcal{C}\delta$. For all $j \in \{0,1\}$, let t_j be as found by (ii).

Define with **KRT** $g = \varphi_e$ so that g works according to the following informal definition in stages. For each s, g_s denotes the finite initial segment of g as defined just before the beginning of stage s.

$g_0 = \varnothing$
`stage` $s = 0$ `to` ∞
  `let` $\tau_0 := t_0(e, g_s)$ `and` $\tau_1 := t_1(e, g_s)$
  `if` $h(g_s\tau_0) \neq h(g_s)$
    `define` $g_{s+1} = g_s \diamond \tau_0$
  `otherwise`
    `define` $g_{s+1} = g_s \diamond \tau_1$
  `goto stage` $s+1$

For $s \in \mathbb{N}$ and $j \in \{0,1\}$ we define

$$\tau_j^s := t_0(e, g_s). \tag{28}$$

Claim 1: We have (29) and (30) just below.

$$\forall s \in \mathbb{N} : (g_s \text{ is defined } \wedge\ g_s \in [\mathcal{S}_e]). \tag{29}$$

$$\forall s \in \mathbb{N} : (\tau_0^s\downarrow\ \wedge\ \tau_1^s). \tag{30}$$

Proof. We show the claim by induction on s with trivial base case. Let now $s \in \mathbb{N}$ such that the claim holds for s. By (iia) we have $\tau_0^{s+1}\downarrow$ and $\tau_1^{s+1}\downarrow$, as well as $h(g_s \diamond \tau_0^s) \neq h(g_s)$ is a computable predicate. We use (iic) and (iid) to see that g_{s+1} is defined and $g_{s+1} \in [\mathcal{S}_e]$. □ (for Claim 1)

By Claim 1, all stages will be reached. Furthermore, by (iib), for all s, $\#\mathrm{elets}(\tau_0^s), \#\mathrm{elets}(\tau_1^s) > 0$; hence, $g(i)$ will be defined no later then after stage i. Thus,

$$g \in \mathcal{R}. \tag{31}$$

By (31), (i) and Claim 1, we now have $g \in \mathcal{S}$.

Claim 2: h does not converge on g.
Proof. We show that, for any stage s, there exist a $y \geq \#\mathrm{elets}(g_s)$ such that $h(g[y+1]) \neq h(g[y])$. Let s be any stage.
Case 1: $h(g_s \diamond \tau_0^s) \neq h(g_s)$. Trivial.
Case 2: $h(g_s \diamond \tau_0^s) = h(g_s)$ and $h(g_s \diamond \tau_1^s) \neq h(g_s)$. Trivial.
Case 3: $h(g_s \diamond \tau_0^s) = h(g_s) = h(g_s \diamond \tau_1^s)$.
$\mathcal{S}$, we have $(h \circ \overline{g[x_s]\tau_0}, g[x_s]\tau_0), (h \circ \overline{g[x_s]\tau_1}, g[x_s]\tau_1) \in [\delta]$. By (iie), we have that $\varphi_{h(g_s \diamond \tau_1^s)} \neq g$. As $\delta \subseteq \mathbf{Ex}$ and h δ-learns $g \in \mathcal{S}$, there is a $y \geq \#\mathrm{elets}(g_s)$ as required.
□ (for Claim 2) □ (for Lemma)

4.1 Results mostly not Comparing Graphs

The following theorem shows the relationship between the different learning criteria as defined in this paper.

Theorem 4.3. *We have the following.*

$$\forall G \in \mathcal{G}_{\text{comp}} : \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex} = \mathcal{T}\mathbf{Pcs}_G\mathbf{Ex}. \tag{32}$$

$$\mathcal{R}\mathbf{Pcs}_\varnothing\mathbf{Ex} \setminus (\bigcup_{G\in\mathcal{G}} \mathbf{Pcp}_G\mathbf{Ex}) \neq \varnothing. \tag{33}$$

$$\mathcal{R}\mathbf{Pcp}_\varnothing\mathbf{Ex} \setminus (\bigcup_{G\in\mathcal{G}} \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}) \neq \varnothing. \tag{34}$$

$$\mathbf{Pcp}_\varnothing\mathbf{Ex} \setminus (\bigcup_{G\in\mathcal{G}_{\text{comp}}} \mathcal{R}\mathbf{Pcs}_G\mathbf{Ex}) \neq \varnothing. \tag{35}$$

Furthermore, the separations (33) and (34) are witnessed by sets of functions such that the positive part of the separation is witnessed by a (fair) learner computable in linear time working transductively.

Proof of (32). This proof of (32) above is an extension of Fulk's proof of the $G = \varnothing$ case [Ful88]. Let $G \in \mathcal{G}_{\text{comp}}$.
"$\subseteq$": Clear.
"$\supseteq$": Let $\mathcal{S} \in \mathcal{T}\mathbf{Pcs}_G\mathbf{Ex}$ as witnessed by $\langle h, f\rangle \in \mathcal{T}\mathbf{Pcs}_G$. Let R be a computable predicate such that, for all p, p', n, τ, t,

$$\begin{aligned} &R(p,p',n,\tau,t) \text{ iff } [p' = h(\tau) \ \wedge \ n \in \perp(f,\tau) \wedge \\ &\forall n' < n : (n' \in \perp(f,\tau) \Rightarrow (\varphi_p(n')\downarrow = \tau(n') \text{ in} \leq t \text{ steps}))]. \end{aligned}$$

By padded **PKRT** there is a 1-1 $e \in \mathcal{R}$ such that

$$\forall \sigma, n : \varphi_{e(\sigma)}(n) = \begin{cases} \sigma(n), & \text{if } n \in \operatorname{dom}(\sigma); \\ \tau_0(n), & \text{else if } \exists \langle \tau_0, t_0\rangle = \\ & \mu\langle \tau, t\rangle . R(e(\sigma), h(\sigma), n, \tau, t); \\ \uparrow, & \text{otherwise.} \end{cases} \tag{36}$$

Let S be a partial computable predicate such that, for all p, p', n, σ,

$$\begin{aligned} S(p,p',n,\sigma) \text{ iff } [&p' = h(\sigma) \wedge \exists t : R(p,p',n,\sigma,t) \\ &\text{and with } \langle \tau_0, t_0\rangle = \mu\langle \tau, t\rangle . R(p,p',n,\tau,t) \\ &\text{we have } \sigma(n) = \tau_0(n)]. \end{aligned} \tag{37}$$

Define h' such that, for all σ,

$$h'(\sigma) = \begin{cases} & \text{if } \sigma \neq \varnothing \wedge h(\sigma) = h(\sigma^-) \wedge \\ h'(\sigma^-), & \text{with } \sigma_0 \text{ such that} \\ & h'(\sigma^-) = e(\sigma_0) \colon \ \forall n \in\!\!\perp\!\!\!\perp (f,\sigma) \colon \\ & S(h'(\sigma^-), h(\sigma_0), n, \sigma); \\ e(\sigma), & \text{otherwise.} \end{cases} \tag{38}$$

Claim 1: $h' \in \mathcal{R}$.
We prove $\forall \sigma \, \exists \sigma_0 \, \forall \sigma_1 : \sigma_0 \subseteq \sigma_1 \subseteq \sigma \Rightarrow h'(\sigma)\downarrow = e(\sigma_0)$ by induction on σ, trivial for $\sigma = \varnothing$. Let σ and σ_0 be given such that

$$\forall \sigma_1 : \sigma_0 \subseteq \sigma_1 \subseteq \sigma^- \Rightarrow h'(\sigma_1)\downarrow = e(\sigma_0). \tag{39}$$

Claim 1.1:

$$\forall n : (\exists \tau : S(e(\sigma_0), h(\sigma_0), n, \tau)\downarrow = \text{ true }) \Rightarrow \varphi_{e(\sigma_0)}(n)\downarrow. \tag{40}$$

Proof of Claim 1.1. Let $n \in \mathbb{N}$ such that $\exists \tau : S(e(\sigma_0), h(\sigma_0), n, \tau)\downarrow = \text{true}$. By the definition of S, there are now τ, t such that $R(e(\sigma_0), h(\sigma_0), n, \tau, t)$. The definition of $e(\sigma_0)$ shows that $\varphi_{e(\sigma_0)}(n)\downarrow$.
□ (FOR CLAIM 1.1)

Claim 1.2:

$$\forall n \in \perp(f, \sigma^-) : \varphi_{e(\sigma_0)}(n)\downarrow. \tag{41}$$

Proof of Claim 1.2. Let $n \in \bot(f, \sigma^-)$. If $n < \#\text{elets}(\sigma_0)$, then, trivially, $\varphi_{e(\sigma_0)}(n)\downarrow$. Suppose now $n \geq \#\text{elets}(\sigma_0)$. By choice of n and the definition of $\perp\!\!\!\perp$, there is then σ_1 such that $\sigma_0 \subset \sigma_1 \subset \sigma$ and $n \in \perp\!\!\!\perp(f, \sigma_1)$. From (39) we have that in the definition of $h'(\sigma_1)$ the first case holds. Thus, $S(e(\sigma_0), h(\sigma_0), n, \sigma_1)\downarrow =$ true. By (40) we have $\varphi_{e(\sigma_0)}(n)\downarrow$.

□ (FOR CLAIM 1.2)

Obviously, it suffices now to show the following claim.
Claim 1.3: For all $n \in \perp\!\!\!\perp(f, \sigma)$,

$$(\forall n' \in \perp\!\!\!\perp(f, \sigma) : n' < n \Rightarrow S(e(\sigma_0), h(\sigma_0), n', \sigma)\downarrow = \text{ true}) \Rightarrow S(e(\sigma_0), h(\sigma_0), n', \sigma)\downarrow. \tag{42}$$

Proof of Claim 1.3. Let $n \in \perp\!\!\!\perp(f, \sigma)$ be such that the antecedent of (42) holds. Using (40) and (41) we now have

$$\forall n' \in \bot(f, \sigma) : n' < n \Rightarrow \varphi_{e(\sigma_0)}(n')\downarrow = \sigma(n). \tag{43}$$

We show $S(e(\sigma_0), h(\sigma_0), n, \sigma)\downarrow$. If we can show the second conjunct of S(...) to hold, then the minimization in the third conjunct of S will also terminate. Hence, it remains to show $\exists t R(e(\sigma_0), h(\sigma_0), n, \sigma, t)$. $h(\sigma_0) = h(\sigma)$ and $n \in \bot(f, \sigma)$ are clear, the remainder follows by (43).

□ (FOR CLAIM 1.3) □ (FOR CLAIM 1)

Claim 2: $\langle h', f\rangle \in \mathcal{T}\mathbf{Pcp}_G$.
Let $\sigma, \sigma_0 \in \mathbb{S}\text{eq}$ be such that $h'(\sigma) = e(\sigma_0)$. Obviously, using induction, it now suffices to show $\forall n \in \perp\!\!\!\perp(f, \sigma) : \varphi_{e(\sigma_0)}(n)\downarrow = \sigma(n)$. Let $n \in \perp\!\!\!\perp(f, \sigma)$. We have $S(e(\sigma_0), h(\sigma_0), n, \sigma)\downarrow =$true. From the definitions of S and $e(\sigma_0)$ we now see $\varphi_{e(\sigma_0)}(n) = \sigma(n)$, as both minimizations give the same result.

□ (FOR CLAIM 2)

Claim 3: $\langle h', f\rangle$ witnesses $\mathcal{S} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$.
Let $g \in \mathcal{S}$. There exists $\sigma_\downarrow \subset g$ minimal such that $\forall \sigma : \sigma_\downarrow \subseteq \sigma \subset g \Rightarrow h(\sigma) = h(\sigma_\downarrow)$ and $\varphi_{h(\sigma_\downarrow)} = g$.

We proceed by showing $\forall \sigma : \sigma_\downarrow \subset \sigma \subset g \Rightarrow h'(\sigma) = h'(\sigma_\downarrow)$. Let σ be such that $\sigma_\downarrow \subset \sigma \subset g$. Obviously, using induction, it suffices to show that $h'(\sigma)$ is defined according to the first case. Let σ_0 be such that $h'(\sigma^-) = e(\sigma_0)$. Note that, by the second conjunct in the cases for (38) and because of the minimality of $\sigma_\downarrow$, $\sigma_\downarrow \subseteq \sigma_0$; hence,

$$h(\sigma) = h(\sigma_\downarrow) = h(\sigma_0). \tag{44}$$

Let $n \in \perp\!\!\!\perp(f, \sigma)$. We show $S(e(\sigma_0), h(\sigma_0), n, \sigma)$. By (44), we have $h(\sigma_0) = h(\sigma)$. As shown in the proof of Claim 1, $\exists t : R(e(\sigma_0), h(\sigma_0), n, \sigma, t)$. Then the minimization in the definition of $S(e(\sigma_0), h(\sigma_0), n, \sigma)$ will terminate. Let $\langle \tau_0, t_0\rangle := \mu\langle \tau, t\rangle . R(e(\sigma_0), h(\sigma_0), n, \tau, t)$. By definition of R we have now $h(\tau_0) = h(\sigma_0) \underset{(44)}{=} h(\sigma_\downarrow)$; hence,

$$\varphi_{h(\tau_0)}(n) = \varphi_{h(\sigma_\downarrow)}(n) = g(n)\downarrow;$$

therefore, $\varphi_{h(\tau_0)}(n)\downarrow$, and, by postdictive consistency, $\tau_0(n) = \varphi_{h(\tau_0)}(n) = g(n) = \sigma(n)$.

□ (FOR CLAIM 3) □ (FOR (32))

Proof of (33). Let $\mathcal{S} := \{g \in \mathcal{R} \mid (\overline{0}\diamond(\pi_1 \circ g), g) \in \mathbf{Pcs}_\emptyset\mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{LinFTdPcs}_\emptyset\mathbf{Ex} \subseteq \mathcal{R}\mathbf{Pcs}_\emptyset\mathbf{Ex}$. Let $G \in \mathcal{G}$. We set up to use Lemma 4.2. Suppose by way of contradiction $\mathcal{S} \in \mathbf{Pcp}_G\mathbf{Ex}$, as witnessed by $\langle h, f\rangle$. Define, for all $e \in \mathbb{N}$, $\mathcal{S}_e := \{g \in \mathcal{R} \mid \forall i : \pi_1(\pi_1(g(i))) = e\}$. Note that $[\mathcal{S}_e]$ is uniformly computable in e. Define $t \in \mathcal{P}$ by setting for all e, σ,

$$t(e, \sigma) = \mu\langle \rho, s\rangle . \sigma \diamond \rho \in [\mathcal{S}_e] \ \wedge \ (h(\sigma)\downarrow \neq h(\sigma \diamond \rho)\downarrow \text{ each in} \leq s \text{ steps}).$$

Claim: Let $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e]$, such that $\varphi_e = \sigma$. Then $t(e, \sigma)\downarrow$.
Let, for each $j \in \{0, 1\}$,

$$\begin{aligned} n_j &:= \overline{\langle\langle e, 0\rangle, j\rangle}; \\ i_j &:= \mu i > 0 . \#\text{elets}(\sigma) \in \bot(f, \sigma \diamond n_j^i); \\ \rho_j &:= n_j^{i_j}. \end{aligned}$$

Note that i_j may not be defined and when defined not algorithmically extractable from e, σ and j, as $\bot_G$ not necessarily computable (since G is not necessarily computable). For all $j \in \{0, 1\}$ and $i \in \mathbb{N}$ we have

$\sigma \diamond n_j^i \in [\mathcal{S}]$, as $\varphi_e = \sigma$; thus, as $\langle h, f\rangle$ $\mathbf{Pcp}_G\mathbf{Ex}$-learns $\mathcal{S}$, $\langle h, f\rangle(\sigma \diamond n_j^i)\downarrow$. Hence, for each $j \in \{0,1\}$, i_j and ρ_j are well defined. We have now

$$\varphi_{h(\sigma\diamond\rho_0)}(\#\text{elets}(\sigma))\downarrow = \rho_0(0) \neq \rho_1(0) = \varphi_{h(\sigma\diamond\rho_1)}(\#\text{elets}(\sigma))\downarrow;$$

thus, $h(\sigma \diamond \rho_0) \neq h(\sigma \diamond \rho_1)$ and $t(e,\sigma)\downarrow$. □ (FOR CLAIM)

By setting $t_0 := t_1 := t$, we can now use Lemma 4.2 to show (33). □ (FOR (33))

Proof of (34). [24] Let $\mathcal{S} := \{g \in \mathcal{R} \mid (\overline{0} \diamond (\pi_1 \circ g), g) \in \mathbf{Pcp}_\emptyset\mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{LinFTdPcp}_\emptyset\mathbf{Ex} \subseteq \mathcal{R}\mathbf{Pcp}_\emptyset\mathbf{Ex}$. Let $G \in \mathcal{G}$. We set up to use Lemma 4.2. Suppose by way of contradiction $\mathcal{S} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$ as witnessed by $\langle h, f\rangle$. Define, for all $e \in \mathbb{N}$, $\mathcal{S}_e := \{g \in \mathcal{R} \mid \forall i : \pi_1(\pi_1(g(i))) = e\}$. Note that $[\mathcal{S}_e]$ is uniformly computable in e. Define $t \in \mathcal{P}$ by setting for all e, σ,

$$t(e,\sigma) = \mu\rho.\sigma \diamond \rho \in [\mathcal{S}_e] \wedge h(\sigma) \neq h(\sigma \diamond \rho).$$

Claim: Suppose $e \in \mathbb{N}, \sigma \in \mathbb{S}\text{eq}$. Then $t(e,\sigma)\downarrow$.
Let, for each $j \in \{0,1\}$,

$$\begin{aligned} n_j &:= \overline{\langle\langle e, 0\rangle, j\rangle}; \\ i_j &:= \mu i > 0.\#\text{elets}(\sigma) \in \perp(f, \sigma \diamond n_j^i); \\ \rho_j &:= n_j^{i_j}. \end{aligned}$$

Note that i_j may not be defined and when defined not algorithmically extractable from e, σ and j, as $\perp_G$ not necessarily computable (since G is not necessarily computable). As $\langle h, f\rangle \in \mathcal{T}\mathbf{Pcp}_G$, we have that, for each $j \in \{0,1\}$, i_j and ρ_j are defined and we have

$$\varphi_{h(\sigma\diamond\rho_0)}(\#\text{elets}(\sigma))\downarrow = \rho_0(0) \neq \rho_1(0) = \varphi_{h(\sigma\diamond\rho_1)}(\#\text{elets}(\sigma))\downarrow;$$

thus, $h(\sigma \diamond \rho_0) \neq h(\sigma \diamond \rho_1)$; therefore, as $\sigma \diamond \rho_0, \sigma \diamond \rho_1 \in [\mathcal{S}_e]$, $t(e,\sigma)\downarrow$. □ (FOR CLAIM)

By setting $t_0 := t_1 := t$, we can now use Lemma 4.2 to show $\mathcal{S} \notin \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$, a contradiction.
□ (FOR (34))

Proof of (35). Let $\mathcal{S} := \{g \in \mathcal{R} \mid (\overline{0} \diamond \lambda n.\langle\varphi_{g(n)}(0), 0\rangle, g) \in \mathbf{Pcp}_\emptyset\mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{Pcp}_\emptyset\mathbf{Ex}$. Let $G \in \mathcal{G}_{\text{comp}}$. We set up to use Lemma 4.2. Suppose now, by way of contradiction, $\mathcal{S} \in \mathcal{R}\mathbf{Pcs}_G\mathbf{Ex}$, as witnessed by

$$\langle h, f\rangle \in \mathcal{R}. \tag{45}$$

By padded **ORT** there is a 1-1 function $P \in \mathcal{R}$ such that

$$P = \lambda\langle d, j, e, \sigma, n\rangle. = \begin{cases} \ell & \text{if } d = 0; \\ p_{j,e,\sigma}(n) & \text{otherwise,} \end{cases} \tag{46}$$

where ℓ and $p_{j,e,\sigma}$ are defined just below.[25]

$$\forall j, e, \sigma : \varphi_\ell(j, e, \sigma) = \mu i > 0.\#\text{elets}(\sigma) \in \perp(f, \sigma \diamond p_{j,e,\sigma}[i]) \tag{47}$$

$$\forall e, \sigma, n, x : \varphi_{p_{0,e,\sigma}(n)}(x) = \begin{cases} \text{patch}_0(\sigma \diamond p_{0,e,\sigma}[n+1]), & \text{if } h(\sigma) = h(\sigma \diamond p_{0,e,\sigma}[\varphi_\ell(0,e,\sigma)]) \\ e, & \text{otherwise; and} \end{cases} \tag{48}$$

$$\forall n, x \forall j > 0 : \varphi_{p_{j,e,\sigma}(n)}(x) = e. \tag{49}$$

[24] An anonymous referee pointed out that (34) can easily be proven by showing that all sets in $\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$ can be reliably learned, as it is known that not all reliably learnable sets are $\mathcal{R}\mathbf{Pcp}_G\mathbf{Ex}$-learnable [CJSW04]. We retain our original proof of (34) herein, since it exercises, in a simple way, an application of Lemma 4.2.

[25] Recall that the properties of patch_0 are listed in (7-9).

Clearly, as P above is total, and by (46), we have that each function $p_{j,e,\sigma}$ is total. Hence, by (45) and (47) we have that φ_l is total. Therefore, by (48), for all j, e, σ, $\varphi_{p_{j,e,\sigma}}$ is total. For each $j \in \{0,1\}$, define

$$t_j(e,\sigma) := p_{j,e,\sigma}[\varphi_\ell(j,e,\sigma)]. \tag{50}$$

By the discussion before (20) we have for all j,

$$t_j \in \mathcal{R}. \tag{51}$$

Let, for all $e \in \mathbb{N}$, $\mathcal{S}_e := \{g \in \mathcal{R} \mid \forall n \in \mathbb{N} : \varphi_{g(n)}(0) = e\}$. We apply Lemma 4.2 with $\mathcal{C} = \mathcal{R}$ and $\delta = \mathbf{Pcs}_G\mathbf{Ex}$. (i) is trivial. To show (ii), let $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e]$. (a) follows from (45) and (51). The conclusion of (b) is trivial from (50). (c) follows from (47), (48) and (50). (d) is trivial from (49). We show (e) by showing the contrapositive: Suppose $\varphi_{h(\sigma\diamond\tau_1)} \in \mathcal{R}$. Define, for each $j \in \{0,1\}$, $\tau_j := t_j(e,\sigma)$. Note that, as P is 1-1, we have $\tau_0(0) = p_{0,e,\sigma}(0) \neq p_{1,e,\sigma}(0) = \tau_1(0)$. Then, by (47),

$$\varphi_{h(\sigma\diamond\tau_0)}(\#\text{elets}(\sigma)) = \tau_0(0) \neq \tau_1(0) = \varphi_{h(\sigma\diamond\tau_1)}(\#\text{elets}(\sigma)).$$

Thus, $h(\sigma \diamond \tau_0) \neq h(\sigma \diamond \tau_1)$, which shows (iie). Lemma 4.2 gives now $\mathcal{S} \notin \mathcal{R}\mathbf{Pcs}_G\mathbf{Ex}$, a contradiction. □ (FOR (35))

Definition 4.4 ([Min76, BB75]). *Given a set $\mathfrak{F} \subseteq \mathfrak{T}$, a function $\langle h, f\rangle \in \mathcal{R}$ is called* $\mathfrak{F}$-reliable *iff* $\forall g \in \mathfrak{F} : \lambda i.h(g[i])$ *converges* $\Rightarrow \forall^\infty i : \varphi_{h(g[i])} = g$. *The set of all $\mathfrak{F}$-reliable functions is denoted by* $\mathrm{Rel}_{\mathfrak{F}}$.

- $\langle h, f\rangle$ *is said to be* reliable, *iff* $\langle h, f\rangle \in \mathrm{Rel}_{\mathcal{R}}$.
- $\langle h, f\rangle$ *is said to be* monotonically reliable, *iff $\langle h, f\rangle$ is reliable and h is monotone (that is, for all $\sigma \subseteq \tau : h(\sigma) \leq h(\tau)$).*

Let pad : $\mathbb{N}^2 \to \mathbb{N}$ be a 1-1 computable function such that $\forall e, n \in \mathbb{N} : \varphi_e = \varphi_{\mathrm{pad}(e,n)}$ such that $\forall e, n \in \mathbb{N} : \mathrm{pad}(e,n) \geq n$).

Lemma 4.5. *Let $\langle h, f\rangle$ be reliable. Then there is a monotonically reliable h' such that $\forall \sigma : [\forall n \leq \#\text{elets}(\sigma) : h(\sigma[n])\downarrow] \Rightarrow \varphi_{h(\sigma)} = \varphi_{h'(\sigma)}$. Furthermore, a program number for h' can be obtained constructively from a program number for h.*

Proof. Define

$$\forall \sigma : h'(\sigma) = \begin{cases} h(\sigma), & \text{if } \sigma = \varnothing; \\ h'(\sigma^-), & \text{if } \sigma \neq \varnothing \text{ and } h(\sigma) = h(\sigma^-); \\ \mathrm{pad}(h(\sigma), h(\sigma^-)), & \text{otherwise.} \end{cases}$$

□

Theorem 4.6. *Let $G \in \mathcal{G}$. Then $\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$ is closed under computably enumerable unions.*

Our proof for Theorem 4.6 makes use of the notion of *reliability* [Min76, BB75].

Proof. Suppose, for each $i \in \mathbb{N}$,

$$\langle h^i, f^i\rangle \text{ witnesses } \mathcal{S}_i \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}, \tag{52}$$

such that $\lambda i, \sigma.\langle h^i(\sigma), f^i(\sigma)\rangle$ is computable. It is easy to see that, for all $i \in \mathbb{N}$, $\langle h^i, f^i\rangle$ is reliable. By padding [Rog67] we can then assume without loss of generality $\langle h^i, f^i\rangle$ is also monotonically reliable. assume all h^i to be monotonically reliable.

Define $i, n, h^\infty, f^\infty \in \mathcal{R}$ such that, for all σ and for all $k < \#\text{elets}(\sigma)$,

$$i(\sigma) = \mu j \leq \#\text{elets}(\sigma).(h^j(\sigma) = \min_{\ell \leq \#\text{elets}(\sigma)} h^\ell(\sigma)); \tag{53}$$

$$n(\sigma) = \left\{ \begin{array}{c} \mu m \leq \#\text{elets}(\sigma).(\forall j | m \leq j < \#\text{elets}(\sigma)) : \\ i(\sigma) = i(\sigma[j]); \end{array} \right. \tag{54}$$

$$h^\infty(\sigma) = \mathrm{patch}(\sigma[n(\sigma)], h^{i(\sigma)}(\sigma)); \tag{55}$$

$$f^\infty(\sigma) = f^{i(\sigma)}(\sigma). \tag{56}$$

Intuitively, i defines which learner to use when seeing σ. n defines the most recent number where i changed the learner to use. $\langle h^\infty, f^\infty\rangle$ is our learner for the union.

Claim 1: $\langle h^\infty, f^\infty\rangle$ works postdictively completely on all $g \in \mathcal{R}$.
Proof. Let $\sigma \in \mathbb{S}\mathrm{eq}$, let $k \in \bot(f, \sigma)$. Let $n_0 := n(\sigma)$.
Case 1: $k < n_0$.
Then we have

$$\varphi_{h^\infty(\sigma)}(k) \underset{(55)}{=} \varphi_{\mathrm{patch}(\sigma[n_0], h^{i(\sigma)}(\sigma))}(k) \underset{k<n_0}{=} \sigma(k).$$

Case 2: $k \geq n_0$.
Then we have

$$\begin{array}{ll} & \mathrm{row}(k, f^\infty, \sigma) \\ \underset{(16)}{=} & \lambda i \leq \#\mathrm{elets}(\sigma) - k \mathbin{.} f^\infty(\sigma[i+k])(\#\mathrm{elets}(\sigma)) \\ \underset{(55)}{=} & \lambda i \leq \#\mathrm{elets}(\sigma) - k \mathbin{.} f^{i(\sigma[i+k])}(\sigma[i+k])(\#\mathrm{elets}(\sigma)) \\ \underset{k\geq n_0}{=} & \lambda i \leq \#\mathrm{elets}(\sigma) - k \mathbin{.} f^{i(\sigma)}(\sigma[i+k])(\#\mathrm{elets}(\sigma)) \\ \underset{(16)}{=} & \mathrm{row}(k, f^{i(\sigma)}, \sigma). \end{array} \tag{57}$$

Hence, we have

$$k \in \bot(f^\infty, \sigma) \underset{(17)}{\Leftrightarrow} \mathrm{row}(k, f^\infty, \sigma) \notin \vec{G} \tag{58}$$

$$\underset{(57)}{\Leftrightarrow} \mathrm{row}(k, f^{i(\sigma)}, \sigma) \notin \vec{G} \tag{59}$$

$$\underset{(17)}{\Leftrightarrow} k \in \bot(f^{i(\sigma)}, \sigma) \tag{60}$$

$$\underset{(52)}{\Rightarrow} \varphi_{h^{i(\sigma)}(\sigma)}(k) = \sigma(k) \tag{61}$$

$$\underset{(55)}{\Leftrightarrow} \varphi_{h^\infty(\sigma)}(k) = \sigma(k). \tag{62}$$

□ (for Claim 1)

Claim 2: h^∞ converges on all $g \in \bigcup_j \mathcal{S}_j$ to a program number for g.
Proof. Let $g \in \bigcup_j \mathcal{S}_j$. Define $M := \{k \mid h^k \text{ converges on } g\}$, $N := \{k \mid h^k \text{ converges on } g \text{ to some } p_k \in \mathbb{N} \wedge (\forall j : h^j \text{ converges on } g \text{ to some } p_j \in \mathbb{N} \Rightarrow p_k \leq p_j)\}$. Obviously, $N \neq \varnothing$.
Claim 2.1: i converges on g to $\min(N)$.
Proof. Let $p \in \mathbb{N}$ be such that $h^{\min(N)}$ converges on g to p. We have, for all $k \notin M$, as h^k is monotonically reliable, $\forall^\infty t : h^k(g[t]) > p$. Similarly, for all $k \in M \setminus N$ we have $\forall^\infty t : h^k(g[t]) > p$. Furthermore, $\forall k \in N \forall^\infty t : h^k(g[t]) = p$. □ (for Claim 2.1)

Now we have that also n converges on g; hence $\langle h^\infty, f^\infty\rangle$ converges on g. □ (for Claim 2)

□ (for Theorem)

Theorem 4.7. *We have*

$$\bigcup_{G\in\mathcal{G}} \mathbf{Pcs}_G\mathbf{Ex} \subset \mathbf{Ex}. \tag{63}$$

Furthermore, the separation is witnessed by a (fair) learner computable in linear time working transductively.

Proof. "$\subseteq$" is trivial.
"$\neq$": Let $\mathcal{S} := \{g \in \mathcal{R} \mid (0 \diamond (\pi_1 \circ g), g) \in \mathbf{Ex}\}$. Obviously, $\mathcal{S} \in \mathbf{LinFTdEx} \subseteq \mathbf{Ex}$. Suppose, by way of contradiction, there are $G \in \mathcal{G}$ and $\langle h, f\rangle \in \mathcal{P}$ such that $\langle h, f\rangle$ witnesses $\mathcal{S} \in \mathbf{Pcs}_G\mathbf{Ex}$. Note that

$$[\mathcal{S}] = \mathbb{S}\mathrm{eq}. \tag{64}$$

Hence, $\langle h, f\rangle \in \mathcal{R}$. We set up to use Lemma 4.2. Let, for all e, $\mathcal{S}_e := \{g \in \mathcal{R} \mid \forall i : \pi_1(\pi_1(g(i))) = e\}$. $[\mathcal{S}_e]$ is uniformly computable in e. Define $t \in \mathcal{P}$ such that

$$\forall e, \sigma : t(e, \sigma) = \mu\tau \mathbin{.} (\sigma \diamond \tau \in [\mathcal{S}_e] \wedge h(\sigma) \neq h(\sigma \diamond \tau)). \tag{65}$$

Claim: For all $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e]$, $t(e,\sigma)\downarrow$.
Proof. Suppose, by way of contradiction, there are $e \in \mathbb{N}, \sigma \in [\mathcal{S}_e]$ such that $t(e,\sigma)\uparrow$. Hence,

$$\forall \tau : \sigma \diamond \tau \in [\mathcal{S}_e] \Rightarrow h(\sigma) = h(\sigma \diamond \tau). \tag{66}$$

Obviously, there are τ, τ' such that $\sigma \diamond \tau, \sigma \diamond \tau' \in [\mathcal{S}_e]$, $\#\text{elets}(\sigma) \in \bot(f, \sigma \diamond \tau)$, $\#\text{elets}(\sigma) \in \bot(f, \sigma \diamond \tau')$ and $\tau(0) \neq \tau'(0)$. Hence, as $\sigma \diamond \tau, \sigma \diamond \tau' \in [\mathcal{S}_e] \subseteq [\mathcal{S}]$, and $\langle h, f\rangle$ works postdictively consistently on $\mathcal{S}$, we have with (66), $\varphi_{h(\sigma)}(\#\text{elets}(\sigma))\uparrow$. Let $g \in \mathcal{S}_e$ be an extension of σ. By (66), h on g converges to $h(\sigma)$, which is not a program number for g (as $\varphi_{h(\sigma)}$ is not total), a contradiction. □ (FOR CLAIM)

We apply Lemma 4.2 with $t_0 := t_1 := t$, $\mathcal{C} = \mathcal{P}$ and $\delta = \mathbf{Pcs}_G\mathbf{Ex}$. Therefore, $\mathcal{S} \notin \mathbf{Pcs}_G\mathbf{Ex}$, a contradiction. □ (FOR THEOREM)

4.2 Dependencies on the Countdown Graphs

Next we define a pre-order, $\leq_{CD}$, on $\mathcal{G}$. We will see that $\leq_{CD}$ characterizes relative learning-power in dependence on countdown graphs.

Definition 4.8. *For two graphs G, G' we write $G \leq_{CD} G'$ (read: G* is countdown reducible to *G') iff there is a $k \in \mathcal{R}$, such that*

(i) for all $y \in G$: $k(\overline{y}) \in G'$;

(ii) for all $\tau \diamond \overline{y} \in \vec{G}$ such that $\#\text{elets}(\tau) > 0$, we have $k(\tau) \rightarrow_{G'} k(\tau \diamond \overline{y})$.

Intuitively, k maps any G-path into a vertex of G'.[26] Clearly, $\leq_{CD}$ is a pre-order.

Proposition 4.9. *Let $G, G' \in \mathcal{G}$. Let $k \in \mathcal{R}$. The following are equivalent.*

(a) $G \leq_{CD} G'$ as witnessed by k;

(b) $\forall \tau \in \vec{G} : (\lambda i < \#\text{elets}(\tau).k(\tau[i+1])) \in \vec{G'}$.

Next we exhibit nice example countdown graphs and indicate how they compare by $\leq_{CD}$.

Definition 4.10. *We will use the following computability-theoretic notions.*

- *A set $A \subseteq \mathbb{N}$ is called* semi-recursive *iff (by a characterization by McLaughlin and Appel, cited in [Joc68, Theorem 4.1(iii)]) A is an initial segment of some computable linear ordering of the natural numbers.*
- *A set $A \subseteq \mathbb{N}$ is called* immune *iff A is infinite and does not contain a* ce *set [Rog67, § 8.2].*
- *A set $A \subseteq \mathbb{N}$ is called* hyper-immune *iff A is infinite and for the unique $r \in \mathfrak{T}$ strictly monotonic increasing such that $\text{range}(r) = A$ we have $\forall f \in \mathcal{R} \exists x \in \mathbb{N} : f(x) < r(x)$ [Rog67, § 9.5]. Note that every hyper-immune set is immune [Rog67, § 9.5].*

ω denotes the order-type of the natural numbers ordered by $\leq$, ω^{-1} denotes the order-type of the natural numbers ordered by $\geq$.

Theorem 4.11. *There are a computable total ordering $\leq_R$ on $\mathbb{N}$ and a set $A \subseteq \mathbb{N}$ such that A is semi-recursive, A and $\overline{A}$ are hyperimmune, hence immune, $\leq_R|_A$ an initial segment of $\leq_R$, $\leq_R|_A$ is of order-type ω and $\leq_R|_{\overline{A}}$ is of order-type ω^{-1}. In particular, $\leq_R$ is of order-type $\omega + \omega^{-1}$ and there are no computable infinitely descending chains with respect to $\leq_R$; hence, $(\mathbb{N}, >_R)$ is a countdown graph.*

Proof. By [Joc68, Theorem 5.2], there is a semi-recursive, hyper-immune set A, such that $\overline{A}$ is hyper-immune. As A semi-recursive, there exists a computable total ordering $\leq_R$ on $\mathbb{N}$ such that A is an initial segment of this ordering. As A (and $\overline{A}$) are not computable, $\leq_R|_A$ does not have a maximal element, and $\leq_R|_{\overline{A}}$ does not have a minimal element. As A and $\overline{A}$ are both immune, we now have by [Cas76, Lemma 2], that $\leq_R|_A$ is of order-type ω and $\leq_R|_{\overline{A}}$ is of order-type ω^{-1}.
Every infinitely descending chain is therefore a subset of $\overline{A}$. As $\overline{A}$ is immune, these chains are not computable. □

For the rest of this section, let $\leq_R$ be as in Theorem 4.11, and let R denote the countdown graph $(\mathbb{N}, >_R)$.

[26] Neither of mapping G vertices into G' vertices nor mapping G paths into G' paths will give us the same characterization results that we have in Theorem 4.14 below.

Example 4.12. *Let $(\mathcal{N}, \leq_{\mathcal{N}}), (\mathcal{N}', \leq_{\mathcal{N}'})$ be computably related systems of ordinal notations. Then we have*

(a) $\mathcal{N} \leq_{CD} \mathcal{N}' \Rightarrow \mathcal{N}'$ gives a notation to at least all the ordinals $\mathcal{N}$ gives a notation to;

(b) $\mathcal{N} \leq_{CD} R \Leftrightarrow \mathcal{N}$ gives a notation to all and only the ordinals $< \omega \cdot i + j$ for some $i \in \{0,1\}, j \in \mathbb{N}$; and

(c) $R \not\leq_{CD} \mathcal{N}$.

Proof. For all $u \in \mathcal{N}$, define $M_u := \{\tau | \tau \diamond \overline{u} \text{ is an } \mathcal{N}\text{-path}\}$. Clearly, for all $u \in \mathcal{N}$, $M_u \neq \emptyset$.
Proof of (a). Suppose $\mathcal{N} \leq_{CD} \mathcal{N}'$ as witnessed by k. Obviously, it suffices to show the following claim.
Claim: $\forall u \in \mathcal{N}, \forall \tau \in M_u : \nu_{\mathcal{N}}(u) \leq \nu_{\mathcal{N}'}(k(\tau \diamond \overline{u}))$.
Proof of Claim. We prove the claim by transfinite induction on $\nu_{\mathcal{N}}(u)$ for $u \in \mathcal{N}$. The base case is trivial. Suppose $u \in \mathcal{N}$ is such that $\nu_{\mathcal{N}}(u) > 0$ and the claim holds for all $v <_{\mathcal{N}} u$. Let $\tau \in M_u$. For all $v <_{\mathcal{N}} u$ we now have

$$\nu_{\mathcal{N}}(v) \underset{(IH)}{\leq} \nu_{\mathcal{N}'}(k(\tau \diamond \overline{u} \diamond \overline{v})) < \nu_{\mathcal{N}'}(k(\tau \diamond \overline{u})). \tag{67}$$

Thus,

$$\nu_{\mathcal{N}}(u) = \sup_{v <_{\mathcal{N}} u} (\nu_{\mathcal{N}}(v) + 1) \underset{(67)}{\leq} \nu_{\mathcal{N}'}(k(\tau \diamond \overline{u})).$$

□ (FOR CLAIM) □ (FOR (a))

Proof of (b). "⇒": Suppose $\mathcal{N} \leq_{CD} R$ as witnessed by $k \in \mathcal{R}$. Suppose, by way of contradiction, $\mathcal{N}$ gives a notation to all ordinals $< \omega \cdot 2$. Let w be a notation in S for ω. It is straightforward that, for all $\tau \in M_w$, $k(\tau \diamond \overline{w}) \in \overline{A}$. We have that $M := \{\tau^- \mid \forall i < \#\text{elets}(\tau) - 1 : \tau(i+1) \text{ is predecessor of } \tau(i) \wedge \text{last}(\tau) \text{ is a notation for a limit-ordinal}\}$ is a ce subset of M_w. Hence, is a ce subset of $\overline{A}$. As $\overline{A}$ is immune, T is finite. Let n be the cardinality of T. Let $\tau \in M$ be a sequence of length $n+1$. Hence, $\{k(\tau[i+1]) | i \leq n\}$ is a subset of T of size $n+1$, a contradiction.

"⇐": Let $i \in \{0,1\}$, $j \in \mathbb{N}$ and $\mathcal{N}$ systems of ordinal notations having notations for all and only the ordinals $< \omega \cdot i + j$. It is easy to see from the definition of a system of ordinal notations, that there is a computable function $f \in \mathcal{R}$ such that $\forall u \in \mathcal{N} : f(u) = \langle a, b \rangle \Leftrightarrow \nu_{\mathcal{N}}(u) = \omega \cdot a + b$.

Let r be a finite sequence strictly increasing with respect to $\leq_R$ in $\overline{A}$ of length $\max(j, 1)$. As $\mathcal{N}$ computably related, there exists $k \in \mathcal{R}$ such that

$$\forall \tau \in \vec{\mathcal{N}} : k(\tau) := \begin{cases} r(b), & \text{if } f(\tau(\#\text{elets}(\tau) - 1)) = \langle 1, b \rangle \text{ for some } b \in \mathbb{N}; \\ \rho_0(\#\text{elets}(\tau_1)), & \text{otherwise, with } \tau = \tau_0 \diamond \overline{v} \diamond \tau_1, \text{ where } \tau_0 \text{ does not contain any notation for a finite ordinal, } f(v) = \langle 0, b' \rangle \text{ and } \rho_0 = \mu\rho \in \vec{R}.\#\text{elets}(\rho) = b' + 1 \ \wedge \ \forall i \leq b' : \rho(i) <_R r(0). \end{cases}$$

□ (FOR (b))

Proof of (c). Suppose, by way of contradiction, otherwise, as witnessed by k. Let r be an infinite decreasing sequence in $\leq_R$. Then $\lambda i.k(r[i+1])$ is an infinite strictly decreasing sequence in $\mathcal{N}$, a contradiction. □ (FOR (c))

We prove "⇒" of Theorem 4.14 below by using a specific set of self-learning functions $\mathcal{S}$. Each $g \in \mathcal{S}$ will give sufficient information as to how to learn it. Intuitively, in order to learn $\mathcal{S}$ by a learner in $\mathcal{T}\mathbf{Pcp}_G$ for some $G \in \mathcal{G}$, this information has to be checked for correctness before being output (and patched if incorrect). In general, this validation may not be computable, but ce. $\mathcal{S}$ will in fact be defined to be the set of all those g, that not only give sufficient information for learning it, but also give an upper bound on the number of steps that a validation will require.

Next we define the *validation*-predicate. The predicate takes the sequence σ of input seen so far and (computably) decides, whether it will be safe to output $\pi_1(\text{last}(\sigma))$ as hypothesis-multicount, using $\pi_2(\text{last}(\sigma))$ as an upper bound for the number of steps that a validation is attempted.

Definition 4.13. *Let $G \in \mathcal{G}_{\text{comp}}$. Let V_G be the following predicate: For all σ, $V_G(\sigma)$ iff $\sigma = \emptyset$ or $\sigma \neq \emptyset$ and, with $e := \pi_1(\pi_1(\text{last}(\sigma)))$, we have for each $l < \#\text{elets}(\sigma)$: $(\text{row}(l, \pi_2 \circ \pi_1 \circ \sigma) \in \vec{G}$ or $\varphi_e(l) = \sigma(l)$ in $\leq \pi_2(\text{last}(\sigma))$ steps).*

Obviously, for $G \in \mathcal{G}_{\text{comp}}$, V_G is computable.

Theorem 4.14. *Let $G \in \mathcal{G}_{\text{comp}}$, $G' \in \mathcal{G}$. We have*

$$\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex} \subseteq \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex} \Leftrightarrow G \leq_{CD} G'.$$

Proof. "$\Leftarrow$": Suppose $G \leq_{CD} G'$ as witnessed by $k \in \mathcal{R}$. Let $\mathcal{S} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$ as witnessed by $\langle h, f\rangle$. We now set f' such that $\langle h, f'\rangle$ witnesses $\mathcal{S} \in \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex}$.

$$\begin{aligned} \forall \sigma \forall l < \#\text{elets}(\sigma) : f'_l(\sigma) &:= k(\text{row}(l, f, \sigma)); \\ \forall \sigma : f'(\sigma) &:= \langle f'_0(\sigma), \ldots, f'_{\#\text{elets}(\sigma)}(\sigma)\rangle. \end{aligned}$$

"$\Rightarrow$": Suppose $\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex} \subseteq \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex}$. Let

$$\mathcal{S} := \{g \in \mathcal{R} \mid (\overline{0} \diamond (\pi_1 \circ g), g) \in \mathbf{Pcp}_G\mathbf{Ex} \wedge \forall n V_G(g[n])\}. \tag{68}$$

Obviously, $\mathcal{S} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$; therefore, $\mathcal{S} \in \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex}$. Let $\langle h, f\rangle$ be such that

$$\langle h, f\rangle \text{ witnesses } \mathcal{S} \in \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex}. \tag{69}$$

In particular, we have now

$$\langle h, f\rangle \in \mathcal{R}. \tag{70}$$

For each $e \in \mathbb{N}$, define

$$\mathcal{S}_e := \{g \in \mathcal{S} \mid \forall i : \pi_1(\pi_1(g(i))) = e\}. \tag{71}$$

Note that,

$$\forall e : [\mathcal{S}_e] \subseteq [\mathcal{S}]. \tag{72}$$

We set $\max(\varnothing) := 0$. Define, for each $j \in \{0,1\}$, $e \in \mathbb{N}$ and $\sigma, \tau \in \mathbb{S}\text{eq}$,

$$r_j(e, \sigma, \tau) = \langle \lambda i < \#\text{elets}(\tau).\langle\langle e, (\tau(i))^{\#\text{elets}(\sigma)+i}\rangle, \max_{x<\#\text{elets}(\sigma)}(\Phi_e(x)) + j\rangle\rangle_{\mathbb{S}\text{eq}}; \tag{73}$$

Obviously, the i-th component of $r_j(e, \sigma, \tau)$ does not depend on any components of τ besides the i-th. Furthermore, note that for all $j \in \{0,1\}$, $e \in \mathbb{N}$, $\sigma \in [\mathcal{S}_e]$ and $\tau \in \vec{G}$,

$$(\sigma \subseteq \varphi_e \wedge r_j(e, \sigma, \tau)\downarrow) \Rightarrow \sigma \diamond r_j(e, \sigma, \tau) \in [\mathcal{S}_e]. \tag{74}$$

$$t(e, \sigma) = \begin{cases} r_j(e, \sigma, \tau), & \text{if } \langle j, \tau\rangle \text{ is first number found in a dovetailing} \\ & \text{search such that } j \in \{0,1\},\ \tau \in \vec{G} \text{ and} \\ & h(\sigma) \neq h(\sigma \diamond r_j(e, \sigma, \tau)\downarrow); \\ \uparrow, & \text{if no such } \langle j, \tau\rangle \text{ is found.} \end{cases} \tag{75}$$

Claim: $(\exists e \in \mathbb{N} \exists \sigma \in [\mathcal{S}_e] : \varphi_e = \sigma \wedge t(e, \sigma)\uparrow) \Rightarrow G \leq_{CD} G'$.
Proof. Suppose $e \in \mathbb{N}$ and $\sigma \in [\mathcal{S}_e]$ are such that

$$\varphi_e = \sigma \wedge t(e, \sigma)\uparrow. \tag{76}$$

Therefore, we have for all $\tau \in \vec{G}$ and $j \in \{0,1\}$, by $\tau, \tau' \neq \varnothing$, (73) and the first conjunct of (76),

$$r_j(e, \sigma, \tau)\downarrow. \tag{77}$$

Note that we have now, for all $\tau, \tau' \in \vec{G}$, by (73) and (77),

$$r_0(e, \sigma, \tau)(0)\downarrow \neq r_1(e, \sigma, \tau')(0)\downarrow. \tag{78}$$

By (75), the second conjunct of (76) and (77) we have, for all $\tau \in \vec{G}$ and $j \in \{0,1\}$,

$$h(\sigma) = h(\sigma \diamond r_j(e, \sigma, \tau)). \tag{79}$$

Obviously, if $\varphi_{h(\sigma)}(\#\text{elets}(\sigma))$ is defined, it can be at most one element in the set $\{r_j(e, \sigma, \tau)(0) \mid j \in \{0,1\}, \tau \in \vec{G}\}$. Hence, with (78), we can (possibly not constructively) fix $j \in \{0,1\}$ such that

$$\varphi_{h(\sigma)}(\#\text{elets}(\sigma)) \notin \{r_j(e, \sigma, \tau)(0) \mid \tau \in \vec{G}\}. \tag{80}$$

By (69), (79) and (80) we have

$$\forall \tau \in \vec{G} : \#\mathrm{elets}(\sigma) \notin \perp_{G'}(f, \sigma \diamond r_j(e, \sigma, \tau)). \tag{81}$$

By (17), this is equivalent to

$$\forall \tau \in \vec{G} : \mathrm{row}(\#\mathrm{elets}(\sigma), f, \sigma \diamond r_j(e, \sigma, \tau)) \in \vec{G'}. \tag{82}$$

We define $k \in \mathcal{R}$ such that, for all $\tau \in \mathbb{N}$,

$$k(\tau) := \begin{cases} f(\sigma \diamond r_j(e, \sigma, \tau))(\#\mathrm{elets}(\sigma)), & \text{if } \tau \in \vec{G}; \\ 0, & \text{otherwise.} \end{cases} \tag{83}$$

or all $\tau \in \vec{G}$, since for all $i < \#\mathrm{elets}(\tau)$, $\tau[i+1] \in \vec{G}$, we have

$$\begin{aligned} & \lambda i < \#\mathrm{elets}(\sigma).k(\tau[i+1]) \\ \underset{(83)}{=} & \lambda i < \#\mathrm{elets}(\sigma).f(\sigma \diamond r_j(e, \sigma, \tau[i+1]))(\#\mathrm{elets}(\sigma)) \\ \underset{(73)}{=} & \lambda i < \#\mathrm{elets}(\sigma).f(\sigma \diamond (r_j(e, \sigma, \tau)[i+1]))(\#\mathrm{elets}(\sigma)) \\ \underset{(16)}{=} & \mathrm{row}(\#\mathrm{elets}(\sigma), f, \sigma \diamond r_j(e, \sigma, \tau)). \end{aligned} \tag{84}$$

(82), (84) and Proposition 4.9 show $G \leq_{CD} G'$. □ (for Claim)

Suppose, by way of contradiction, $G \not\leq_{CD} G'$. Hence, by the claim,

$$\forall e \forall \sigma \in [\mathcal{S}_e] : \varphi_e = \sigma \Rightarrow t(e, \sigma)\downarrow. \tag{85}$$

We apply Lemma 4.2 with $t_0 := t_1 := t$, $\mathcal{C} := \mathcal{T}\mathbf{Pcp}_{G'}$ and $\delta = \mathbf{Ex}$. (i) is trivial from the definitions and (72). (ii)(a) follows with (70) and (85). (ii)(b), (ii)(c) and (ii)(d) are straight from (74) and (75). Furthermore, by (75), we get directly $t_0(e, \sigma)\downarrow \Rightarrow h(\sigma) \neq h(\sigma \diamond t_0(e, \sigma))$; hence, the antecedent of (ii)(e) is false.

Therefore, $\mathcal{S} \notin \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex}$, a contradiction. □ (for Theorem)

Next are three corollaries to Theorem 4.14 (or its proof). The first two are regarding the other restricted learnability notions of the present paper. The third is our hierarchy theorem for ordinal notations.

First, we observe that the set $\mathcal{S}$ as in (68) in the proof of Theorem 4.14 does depend only on G, not on G'. Therefore, we can give the following strong corollary.

Corollary 4.15. *Let $G \in \mathcal{G}_{\mathrm{comp}}$. We have*

$$\mathcal{T}\mathbf{Pcp}_G\mathbf{Ex} \setminus \bigcup_{\substack{G' \in \mathcal{G}_{\mathrm{comp}} \\ G \not\leq_{CD} G'}} \mathbf{Pcs}_{G'}\mathbf{Ex} \neq \emptyset.$$

Proof. Let $\mathcal{S}$ be as given in (68) in the proof of Theorem 4.14. Let $\mathcal{V} := \{g \in \mathcal{R} \mid \forall^\infty x : f(x) = 0\}$, the set of functions of finite support. Clearly, $\mathcal{V} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$; hence, by the union theorem (Theorem 4.6), we have $\mathcal{S}' := \mathcal{S} \cup \mathcal{V} \in \mathcal{T}\mathbf{Pcp}_G\mathbf{Ex}$. Let $G' \in \mathcal{G}_{\mathrm{comp}}$ such that $G \not\leq_{CD} G'$. The proof of Theorem 4.14 showed $\mathcal{S} \notin \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex}$; hence, $\mathcal{S}' \notin \mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex}$. Since $[\mathcal{S}'] = \mathbb{S}\mathrm{eq}$ and $\mathcal{T}\mathbf{Pcp}_{G'}\mathbf{Ex} \underset{(32)}{=} \mathcal{T}\mathbf{Pcs}_{G'}\mathbf{Ex}$, we have, by the contrapositive of (26) given in Remark 4.1,

$$\mathcal{S}' \notin \mathbf{Pcs}_{G'}\mathbf{Ex}.$$

□

Next is a characterization of the graph dependence of relative learning power for the restricted learning criteria not covered by Theorem 4.14.

Corollary 4.16. *For all $G, G' \in \mathcal{G}_{\text{comp}}$ we have*

$$G \leq_{CD} G' \Leftrightarrow \mathcal{R}\mathbf{Pcp}_G\mathbf{Ex} \subseteq \mathcal{R}\mathbf{Pcp}_{G'}\mathbf{Ex} \tag{86}$$
$$\Leftrightarrow \mathcal{R}\mathbf{Pcs}_G\mathbf{Ex} \subseteq \mathcal{R}\mathbf{Pcs}_{G'}\mathbf{Ex} \tag{87}$$
$$\Leftrightarrow \mathbf{Pcp}_G\mathbf{Ex} \subseteq \mathbf{Pcp}_{G'}\mathbf{Ex} \tag{88}$$
$$\Leftrightarrow \mathbf{Pcs}_G\mathbf{Ex} \subseteq \mathbf{Pcs}_{G'}\mathbf{Ex}. \tag{89}$$

Proof. Obviously, all right-hand-sides are implied by $G \leq_{CD} G'$, just as in the proof of "$\Leftarrow$" of Theorem 4.14. By Corollary 4.15, $G \not\leq_{CD} G'$ implies the negation of each right-hand-side. □

Recall that, from Section 2, for a graph $G \in \mathcal{G}$ and $m \in G$, we ambiguously use m to refer to the countdown-graph $\{n \in G \mid m \to^+ n\}$. For two sets M, N we write $M \# N$ iff $(M \notin N \wedge N \notin M)$.

Corollary 4.17. *Let $(\mathcal{N}, \leq_{\mathcal{N}})$ be a computably related system of ordinal notations. Let $u, v \in \mathcal{N}$. Then we have*

$$u <_{\mathcal{N}} v \quad \Leftrightarrow \quad u <_{CD} v \tag{90}$$
$$\Leftrightarrow \quad \mathcal{T}\mathbf{Pcp}_u\mathbf{Ex} \subset \mathcal{T}\mathbf{Pcp}_v\mathbf{Ex}. \tag{91}$$

Furthermore, if $\mathcal{N}$ gives a notation to at least all ordinals $< \omega \cdot 2$, then

$$\mathcal{T}\mathbf{Pcp}_{\mathcal{N}}\mathbf{Ex} \# \mathcal{T}\mathbf{Pcp}_R\mathbf{Ex}. \tag{92}$$

Proof of (90). "$\Rightarrow$": Suppose $u <_{\mathcal{N}} v$. Hence, considering u and v as graphs, we have $u \subset v$. Then we have $u \leq_{CD} v$ is witnessed by any $k \in \mathcal{R}$ such that $\forall \tau \in \vec{\mathcal{N}} : k(\tau) = \text{last}(\tau)$, so that Theorem 4.14 applies. By Example 4.12(a), $v \not\leq_{CD} u$
"$\Leftarrow$": This follows directly from Example 4.12(a). *Proof of (91).* By Theorem 4.14.
Proof of (92). This follows directly from Theorem 4.14 and Example 4.12. □

References

[ACJS04] A. Ambainis, J. Case, S. Jain, and M. Suraj. Parsimony hierarchies for inductive inference. *Journal of Symbolic Logic*, 69:287–328, 2004.

[AZ07] Y. Akama and T. Zeugmann. Consistent and coherent learning with δ-delay. Technical Report TCS-TR-A-07-29, Hokkaido University, October 2007.

[Bār74] J. Bārzdiņš. Inductive inference of automata, functions and programs. In *Int. Math. Congress, Vancouver*, pages 771–776, 1974.

[BB75] L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

[Cas74] J. Case. Periodicity in generations of automata. *Mathematical Systems Theory*, 8:15–32, 1974.

[Cas76] J. Case. Sortability and extensibility of the graphs of r.e. partial and total orders. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 22:1–18, 1976.

[Cas94] J. Case. Infinitary self-reference in learning theory. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:3–16, 1994.

[CJSW04] J. Case, S. Jain, F. Stephan, and R. Wiehagen. Robust learning – rich and poor. *Journal of Computer and System Sciences*, 69:123–165, 2004.

[CKP07] John Case, Timo Kötzing, and Todd Paddock. Feasible iteration of feasible learning functionals. In Marcus Hutter, Rocco A. Servedio, and Eiji Takimoto, editors, *ALT*, volume 4754 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2007.

[FS93] R. Freivalds and C. Smith. On the role of procrastination in machine learning. *Information and Computation*, 107(2):237–271, 1993.

[Ful88] M. Fulk. Saving the phenomenon: Requirements that inductive machines not contradict known data. *Information and Computation*, 79:193–209, 1988.

[Joc68] C. Jockusch. Semirecursive sets and positive reducibility. *Transactions of the AMS*, 131:420–436, 1968.

[JORS99] S. Jain, D. Osherson, J. Royer, and A. Sharma. *Systems that Learn: An Introduction to Learning Theory*. MIT Press, Cambridge, Mass., second edition, 1999.

[LV97] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, Heidelberg, second edition, 1997.

[Min76] E. Minicozzi. Some natural properties of strong identification in inductive inference. *Theoretical Computer Science*, pages 345–360, 1976.

[Pit89] L. Pitt. Inductive inference, DFAs, and computational complexity. In *Analogical and Inductive Inference, Proceedings of the Second International Workshop (AII'89)*, volume 397 of *Lecture Notes in Artificial Intelligence*, pages 18–44. Springer-Verlag, Berlin, 1989.

[RC94] J. Royer and J. Case. *Subrecursive Programming Systems: Complexity and Succinctness*. Research monograph in *Progress in Theoretical Computer Science*. Birkhäuser Boston, 1994.

[Rog67] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967. Reprinted by MIT Press, Cambridge, Massachusetts, 1987.

[SSV04] A. Sharma, F. Stephan, and Y. Ventsov. Generalized notions of mind change complexity. *Information and Computation*, 189:235–262, 2004.

[Wie76] R. Wiehagen. Limes-erkennung rekursiver funktionen durch spezielle strategien. *Electronische Informationverarbeitung und Kybernetik*, 12:93–99, 1976.

[Wie78] R. Wiehagen. *Zur Theorie der Algorithmischen Erkennung*. PhD thesis, Humboldt University of Berlin, 1978.